TURING 图灵程序设计丛书

[PACKT] PUBLISHING

[斯洛文尼亚] Boštjan Kaluža 著　武传海 译

# Java 机器学习

# Machine Learning in Java

 中国工信出版集团

 人民邮电出版社
POSTS & TELECOM PRESS

# 数字版权声明

作者简介

Boštjan Kaluža

博士，人工智能与机器学习专家，IT运营分析公司Evolven首席数据科学家，主攻机器学习、预测分析、模式挖掘与异常检测，旨在将数据转化为人类可理解的信息与可应用的知识。个人网址：http://bostjankaluza.net。



图灵程序设计丛书

# Java机器学习

Machine Learning in Java

[斯洛文尼亚] Boštjan Kaluža 著
武传海 译

人民邮电出版社
北京

图书在版编目（CIP）数据

Java机器学习 / （斯洛文）博思蒂安·卡鲁扎著 ；
武传海译. -- 北京 : 人民邮电出版社, 2017.9
（图灵程序设计丛书）
ISBN 978-7-115-46680-8

Ⅰ. ①J… Ⅱ. ①博… ②武… Ⅲ. ①JAVA语言—程序设计 Ⅳ. ①TP312.8

中国版本图书馆CIP数据核字(2017)第202589号

## 内 容 提 要

本书介绍如何使用 Java 创建并实现机器学习算法，既有基础知识，又提供实战案例。主要内容包括：机器学习基本概念、原理，Weka、Mahout、Spark 等常见机器学习库的用法，各类机器学习常见任务，包括分类、预测预报、购物篮分析、检测异常、行为识别、图像识别以及文本分析。最后还提供了相关 Web 资源、各种技术研讨会议以及机器学习挑战赛等进阶所需内容。

本书适合机器学习入门者，尤其是想使用 Java 机器学习库进行数据分析的读者。

◆ 著　[斯洛文尼亚] Boštjan Kaluža
译　武传海
责任编辑　陈　曦
责任印制　彭志环

◆ 人民邮电出版社出版发行　北京市丰台区成寿寺路11号
邮编　100164　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京　印刷

◆ 开本：800×1000　1/16
印张：11.5
字数：272千字　2017年 9 月第 1 版
印数：1－3 500册　2017年 9 月北京第 1 次印刷

著作权合同登记号　图字：01-2017-5590号

定价：49.00元

读者服务热线：(010)51095186转600　印装质量热线：(010)81055316

反盗版热线：(010)81055315

广告经营许可证：京东工商广登字 20170147 号

# 前　言

机器学习是人工智能的一个分支，它在算法与数据的协助下，让计算机像人类一样学习和行动。针对给定的数据集，机器学习算法会学习数据的不同属性，并对以后可能遇到的数据属性进行推断。

本书教你如何使用Java创建并实现机器学习算法，既有基础概念的讲解，也有示例供你学习。当然，还会介绍一些常用的机器学习库，如Weka、Apache Mahout、Mallet等。阅读本书后，你将懂得如何为特定问题选择合适的机器学习方法，以及如何比较与评估不同技术的优劣。书中还会讲解性能提升技术，包括输入预处理以及合并不同方法产生的输出。

我们将探讨使用Java库进行机器学习的技术细节，并配有清晰易懂的示例。同时，你还将学习如何准备要分析的数据、如何选择机器学习方法，以及如何衡量流程的有效性。

## 本书内容

**第1章　机器学习应用快速入门**。讲解机器学习的基础知识、常见概念、原理，以及机器学习的应用流程。

**第2章　面向机器学习的Java库与平台**。介绍各种机器学习专用的Java库与平台。你将了解每个库提供的功能，以及可以用于解决哪些问题。涉及的机器学习库有Weka、Java-ML、Apache Mahout、Apache Spark、Deeplearning4j和Mallet。

**第3章　基本算法——分类、回归和聚类**。从最基本的机器学习任务入手，使用小巧又易懂的数据集，介绍分类、回归和聚类的关键算法。

**第4章　利用集成方法预测客户关系**。深入研究一个真实的客户营销数据库，目标是对可能流失以及可进行追加推销与交叉推销的客户进行预测。我们将使用集成方法解决这个问题，并且采用在KDD Cup竞赛中获胜的解决方案。

**第5章　关联分析**。讲解如何使用关联规则挖掘分析共生关系。我们将通过“购物篮分析”了解顾客的购买行为，并讨论如何将这种方法应用到其他领域。

**第6章　使用Apache Mahout制作推荐引擎**。讲解一些基本概念，帮助你了解推荐引擎原理，然后利用Apache Mahout实现两个应用——基于内容的过滤与协同推荐器。

**第7章　欺诈与异常检测**。介绍异常和可疑模式检测的背景，然后讲解两个实际应用——保险索赔欺诈检测与网站流量异常检测。

**第8章　利用Deeplearning4j进行图像识别**。介绍图像识别与基本的神经网络架构，然后讨论如何利用Deeplearning4j库实现各种深度学习架构，以实现对手写体数字的识别。

**第9章　利用手机传感器进行行为识别**。借助传感器数据解决模式识别问题。这一章介绍行为识别过程，讲解如何使用Android设备收集数据，并提出一个分类模型以对日常生活行为进行识别。

**第10章　利用Mallet进行文本挖掘——主题模型与垃圾邮件检测**。讲解文本挖掘的基础知识，介绍文本处理管道，演示如何将其应用于两个实际问题（主题建模与文档分类）。

**第11章　机器学习进阶**。这是全书最后一章，提供关于如何部署模型的实用建议，并进一步给出提示，告诉你去哪里寻找更多资源、资料、场所和技术，以便深入了解机器学习。

## 阅读前提

为了实际运行书中示例，你需要一台安装有JDK的个人计算机。所有能下载的示例与源代码都假定你使用的是支持Maven（一个依赖管理与自动创建工具）与Git（版本控制系统）的Eclipse IDE开发环境。各章示例依赖Weka、Deeplearning4j、Mallet、Apache Mahout等各种库。关于如何获取与安装这些库，会在各章首次用到它们时进行讲解。

## 读者对象

本书为那些想学习如何使用Java机器学习库进行数据分析的人而写。或许你已经对机器学习有了一点了解，但从未用过Java；又或许你懂得一点Java，而在机器学习方面是个新手。不论你属于哪种情况，本书都能让你快速上手，并提供必需的技能，让你能够成功创建、定制，以及在实际生活中部署机器学习应用。如果你懂得一点基本的编程知识以及数据挖掘相关概念会更好，但不要求你必须拥有与数据挖掘程序包相关的开发经验。

## 配套资料

本书专门配有一个在线支持网站（http://machine-learning-in-java.com），从中可以找到所有示例代码、勘误表，以及其他入门资料。

## 排版约定

本书中，你会发现许多不同体例，它们用于区分不同类型的信息。下面给出一些例子，并对其含义进行说明。

正文中的代码、数据库表名、文件夹名、文件名、文件扩展名、路径名、伪URL、用户输入和Twitter用户名显示如下：

“比如，`Bob`拥有`height`、`eye color`、`hobbies`三个属性，对应值依次为`185cm`、`blue`、`climbing`与`sky diving`”。

代码块表示如下：

```
Bob={
height: 185cm,
eye color: blue,
hobbies: climbing, sky diving
}
```

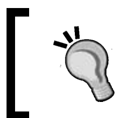
所有命令行输入或输出写成如下形式：

```
12,3,7,2,0,1,8,9,13,4,11,5,15,10,6,14,16
```

**新术语**与**重要词语**使用黑体显示。你在屏幕上看到的词，比如菜单或对话框中的词，在正文中显示如下：“在项目属性上点击鼠标右键，选择**Java Build Path**，单击**Libraries**选项卡，选择**Add External JARs**。”




警告或重要的注意事项。



提示和技巧。

## 读者反馈

欢迎提供反馈，请将你对本书的看法告诉我们：哪些方面是你喜欢的，哪些方面你不喜欢。读者反馈对我们来说很重要，因为这可以帮助我们推出更符合读者需求的著作。

要给我们提供反馈，只需向feedback@packtpub.com发送电子邮件，并在邮件主题中指出书名。

如果你有擅长的主题，并有志于写书或撰稿，请参阅www.packtpub.com/authors的撰稿指南。

# 读者支持

购买本社图书后，你将获得各种帮助，让手中图书最大限度地发挥功效。

## 下载示例代码

你可以从本书配套网站（http://machine-learning-in-java.com）下载书中示例代码。导航到Downloads版块，点击到Git仓库链接。

当然，你也可以使用自己的账号登录http://www.packpub.com网站下载本书示例代码。不论你在哪里购买本书，都可以访问http://www.packtpub.com/support。注册之后，我们会使用电子邮件将示例代码直接发送给你。

下载示例代码时，请按照如下步骤进行：

(1) 使用你的电子邮件地址与密码登录我们的网站，如果尚未加入会员，请先注册加入；
(2) 移动鼠标到SUPPORT菜单之上；
(3) 点击Code Downloads & Errata；
(4) 在Search文本框中输入书名；
(5) 选择你想下载代码文件的图书；
(6) 从下拉菜单中选择你购买本书的地点；
(7) 点击Code Download。

示例文件下载完成后，请使用如下最新版本的解压缩软件对文件进行解压。

- WinRAR/7-Zip for Windows
- Zipeg/iZip/UnRarX for Mac
- 7-Zip/PeaZip for Linux

# 勘误

虽然我们力图让图书内容准确无误，但错误仍不可避免。如果你在本社图书中发现错误（包括正文和代码），请告诉我们，我们将感激不尽。你这样做不仅可以让其他读者免遭同样的挫折，还可帮助我们改进该书的后续版本。无论你发现什么错误，都请告诉我们。为此，可以访问http://www.packtpub.com/submit-errata，输入书名，单击链接Errata Submission Form，再输入错误详情。提交的勘误得到确认后，将被上传到我们的网站或添加到既有的勘误列表。

要查看已提交的勘误，请访问https://www.packtpub.com/books/content/support，并在搜索框中输入书名，Errata栏将列出你搜索的信息。

## 打击盗版

网上散布的盗版材料是各类媒体屡禁不绝的问题。在保护版权和许可方面，本社的态度非常严肃。如果你在网上看到本社作品的非法复制品，请马上把网址或网站名告诉我们，以便我们能够采取补救措施。

请通过copyright@packtpub.com与我们取得联系，并提供可疑的盗版材料链接。

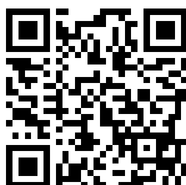
感谢你为保护我们的作者提供的帮助，也十分感激对于我们提供有价值内容的能力给予的保护。

## 答疑

只要有与本书相关的问题，都可通过questions@packtpub.com与我们联系，我们将尽力解决。

## 电子书

扫描如下二维码，即可购买本书电子版。



# 目 录

# 第 1 章 机器学习应用快速入门

本章介绍机器学习的基础知识，包括常见主题与概念，这些内容将让你更容易理解相关逻辑以及所讲主题。本章的目标是让你快速了解**应用机器学习**的详细步骤，掌握机器学习的主要原理。本章涵盖以下内容：

- 介绍机器学习及其与数据科学的关系
- 讨论机器学习应用的基本步骤
- 讨论所处理数据的类型及其重要性
- 讨论收集数据以及对数据进行预处理的方法
- 使用机器学习理解数据
- 使用机器学习从数据获取有用信息并创建预测器

如果你已经熟悉机器学习，并急于开始编写代码，请跳过本章内容，直接阅读其他章节。然而，如果你想重温这些内容或者搞清一些概念，强烈建议你认真学习本章。

## 1.1 机器学习与数据科学

如今，每个人都在谈论机器学习与数据科学。那么，机器学习究竟是什么？它与数据科学有着怎样的联系呢？这两个术语常被混淆，因为它们经常使用相同的方法，有着明显的重叠。因此，下面先区分这两个术语。

Josh Wills在Twitter上说：

> “所谓的数据科学家指这样一类人，他们比软件工程师更懂统计学，比统计学家更懂软件工程。”

更具体地说，数据科学包含从数据获取知识的整个过程，它综合运用统计学、计算机科学以及其他领域的各种方法，帮助人们从数据中获取有用的知识与信息。事实上，数据科学是一个不断反复的过程，包括数据的采集、清洗、分析、可视化和部署。

另一方面，机器学习主要涉及数据科学的分析与建模阶段使用的通用算法与技术。对于机器学习，Arthur Samuel在1959年提出如下定义：

> “机器学习是指研究、设计与开发某些算法，让计算机获得学习的能力，而不需要明确的编程。”

### 1.1.1　机器学习能够解决的问题

机器学习方法主要有如下三种：

- 监督学习
- 无监督学习
- 强化学习

给定一组样本输入$X$与它们的结果$Y$，监督学习的目标是产生一个通用的映射函数$f$，使得每一个输入都有对应的确定输出，即$f$: $X$ $Y$。

监督学习的一个应用例子是检测信用卡欺诈。学习算法会学习所有带有“正常”或“可疑”标记（向量$Y$）的信用卡交易（矩阵$X$），并最终产生一个决策模型（即$f$函数），对未见过的交易打标记（“正常”或“可疑”）。

相反，无监督学习算法所学的数据没有给定的结果标签Y，它主要学习数据的结构，比如将相似的输入数据归入某个聚类。因此，使用无监督学习能够发现隐藏在数据中的模式。无监督学习的一个应用例子是基于物品（Item-based）的推荐系统，其学习算法会发现购物者一同购买的相似商品，比如购买了书A的人也购买了书B。

强化学习从完全不同的角度处理学习过程。它假设有一个智能体（agent，可以是机器人、自动程序或计算机程序）与动态环境进行交互，以实现某个特定目标。环境由一组状态描述，智能体可以做出不同行为，以从一种状态变为另一种状态。有些状态被标为目标状态，如果智能体实现了这种状态，就会获得很大的奖励；而在其他状态中，智能体得到的奖励很少或没有，甚至还会被“惩罚”。强化学习的目标是找到最优策略，即映射函数，指定每个状态要采取的行为动作，而没有指导者（teacher）明确告知这样做是否会实现目标状态。强化学习的一个例子是汽车自动驾驶程序，其中“状态”与“驾驶条件”（比如当前速度、路况信息、周围交通状况、速度限制、路障）相对应，行为会驱动汽车做出诸如左转、右转、停止、加速、前行等动作。学习算法会产生一个策略，指定汽车在特定驾驶条件下要采取的动作。

本书中，我们把学习重点放在监督学习与无监督学习上，因为它们有许多相同的概念。如果强化学习激起了你的兴趣，建议阅读Richard S. Sutton与Andrew Barto二人合写的*Reinforcement Learning: An Introduction*，它是一本很好的入门书。

### 1.1.2　机器学习应用流程

本书讲解的重点是机器学习的应用。我们将提供切实可用的技能，帮助你顺利地将学习算法应用于各种不同的情景设置。相比于机器学习相关的理论与数学知识，我们将把更多时间用来学习如何将机器学习技术应用于具体实践。借助这些实际应用技术，你可以让机器学习在具体应用中发挥强大作用。我们把讲解重点放在监督学习与无监督学习上，涵盖从数据科学到创建机器学习应用流程的所有必需步骤。

标准的机器学习应用流程就是回答一系列问题，可概括成如下5个步骤。

(1) **数据与问题定义**：第一步是要问一些有趣的问题。你试图解决的问题是什么？它为何重要？哪种形式的结果能够回答你的问题？回答是简单的“是与否”吗？你需要从现有问题中挑选吗？

(2) **数据收集**：一旦有问题要解决，你就需要使用相关数据。问一问自己，哪种数据能够帮助你回答问题。你能从现有可用来源获取所需数据吗？你必须对多个来源进行合并吗？你必须生产数据吗？存在取样偏差吗？你需要多少数据？

(3) **数据预处理**：数据预处理的第一项任务是数据清洗，比如填补缺失值、平整噪声数据、移除异常值、解决数据一致性。通常，随后会有对多个数据源的整合以及数据转换，包括把数据转换到特定范围（数据标准化）、将数据分成一系列值段（数据离散化）、降低数据维数。

(4) **利用无监督学习与监督学习进行数据分析与建模**：数据分析与建模包括无监督机器学习与监督机器学习、统计推断和预测。这个阶段有多种机器学习算法可供选用，比如k最近邻算法、朴素贝叶斯算法、决策树、支持向量机、逻辑回归、K均值算法等。至于选用哪种算法，取决于第一步中提到的问题定义以及所收集的数据类型。经过这一步，我们最终会从数据推导出模型。

(5) **模型评价**：最后一步是对模型进行评价。通过机器学习创建模型后，接下来检查模型对源数据的拟合程度。如果模型的针对性太强，即对训练数据过拟合，那么它对新数据的预测效果就很有可能比较差；如果模型太通用，这意味着模型对训练数据欠拟合。比如，向欠拟合的天气预测模型询问加利福尼亚州的天气时，得到的回答总是“晴朗”。大多数时候这个回答是对的，但就有效预测天气来说，这个模型真的没什么用。这一步的目标是准确评价模型，确保模型面对新数据也能正常工作。进行模型评价时，常用的方法有独立测试、训练集、交叉验证、留一法交叉验证。

接下来，我们将详细讲解每个步骤，并且尝试理解机器学习应用流程过程中必须回答的问题类型，还要了解与数据分析、评估相关的概念。

## 1.2　数据与问题定义

简单地说，数据就是一系列测量值，表现形式多样，比如数值、文字、测量值、观测值、事物描述、图像等。

### 测量尺度

数据最常见的表示方式是使用一组属性值对。请看如下例子：

```
Bob={
height: 185cm,
eye color: blue,
hobbies: climbing, sky diving
}
```

Bob拥有height、eye color、hobbies三个属性，它们对应的值依次为185cm、blue、climbing, sky diving。

上面这组数据可以简单表示如下表。表格的列对应属性或特征，表格的行对应特定的数据样本或实例。监督机器学习中，属性代表类或者目标变量，其值是我们想从其他属性值（$X$）进行预测得到的结果（$Y$），如表1-1所示。

表1-1　示例列表

| 姓　　名 | 身高[cm] | 眼球颜色 | 兴趣爱好 |
|---|---|---|---|
| Bob | 185.0 | Blue | Climbing, sky diving |
| Anna | 163.0 | Brown | Reading |
| ... | ... | ... | ... |

我们首先注意到的是属性值的类型。比如，身高是一个数值，眼球颜色是一个文本，兴趣爱好是一个列表。为了更好地理解属性值的类型，下面详细了解数据或测量尺度的不同类型。Stevens（1946）定义了如下4种测量尺度，它们的表现属性在不断增加。

- **称名数据（Nominal data）**相互排斥，但不分顺序，比如眼球颜色、婚姻状况、汽车的品牌等。
- **顺序数据（Ordinal data）**是数据顺序有意义的分类数据，但值之间没有区别，比如疼痛程度、学习成绩字母等级、服务质量评级、IMDB电影评分等。
- **等距数据（Interval data）**中两个值之间的不同具有意义，但是无“绝对0”概念。比如标准化后的考试分数、华氏温度等。
- **等比数据（Ratio data）**拥有等距变量（interval variable）的所有属性，并且还有明确的“0点”定义。变量为0时，表示该变量代表的某种事物或特征不存在。身高、年龄、股票价格、每周伙食支出等都是等比变量。

我们为什么要关注测量尺度呢？机器学习在很大程度上依赖于数据的统计属性，因此应该注意每个数据类型自身具有的限制。有些机器学习算法只能被应用到测量尺度的一个子集上。

表1-2总结了每种测量类型的主要操作与统计特性。

表1-2　不同测量类型的主要操作与统计特性

| 特　性 | 称　名 | 顺　序 | 等　距 | 等　比 |
|---|---|---|---|---|
| 频率分布 | √ | √ | √ | √ |
| 中位数和众数 | | √ | √ | √ |
| 值顺序已知 | | √ | √ | √ |
| 每个值之间的不同可以量化 | | | √ | √ |
| 值可以加减 | | | √ | √ |
| 值可以乘除 | | | | √ |
| 拥有真0点 | | | | √ |

此外，称名数据与顺序数据对应于离散值，而等距数据与等比数据还可以对应于连续值。监督学习中，我们想要预测的属性值的测量尺度决定哪种机器算法可用。例如，从有限列表预测离散值称为“分类”，它可以使用决策树算法实现；而预测连续值称为“回归”，可以使用模型树算法实现。

## 1.3　数据收集

那么，数据从何而来呢？我们有两种选择：从现有源观察并获取数据，或者通过调查、模拟、实验生成数据。下面详细学习这两种方法。

### 1.3.1　发现或观察数据

我们可以从很多地方发现或观察到数据，互联网就是最明显的数据源。英特尔（2013）给出图1-1，显示通过不同互联网服务收集的海量数据。2013年，数字设备产生了4泽字节（1021=10亿太字节）的数据。2017年，入网设备数量预计达到地球总人数的3倍，这些设备产生与收集的数据量将进一步增加。

有多种方法可以从互联网获取数据。

- 从维基百科、IMDb、Million Song Dataset等网站批量下载。
- 通过API（《纽约时报》、推特、脸书、Foursquare）访问数据。
- 网页抓取：从网页上抓取公开、非敏感、匿名数据是可行的。抓取之前，请认真阅读版权条款与完整的引用信息。

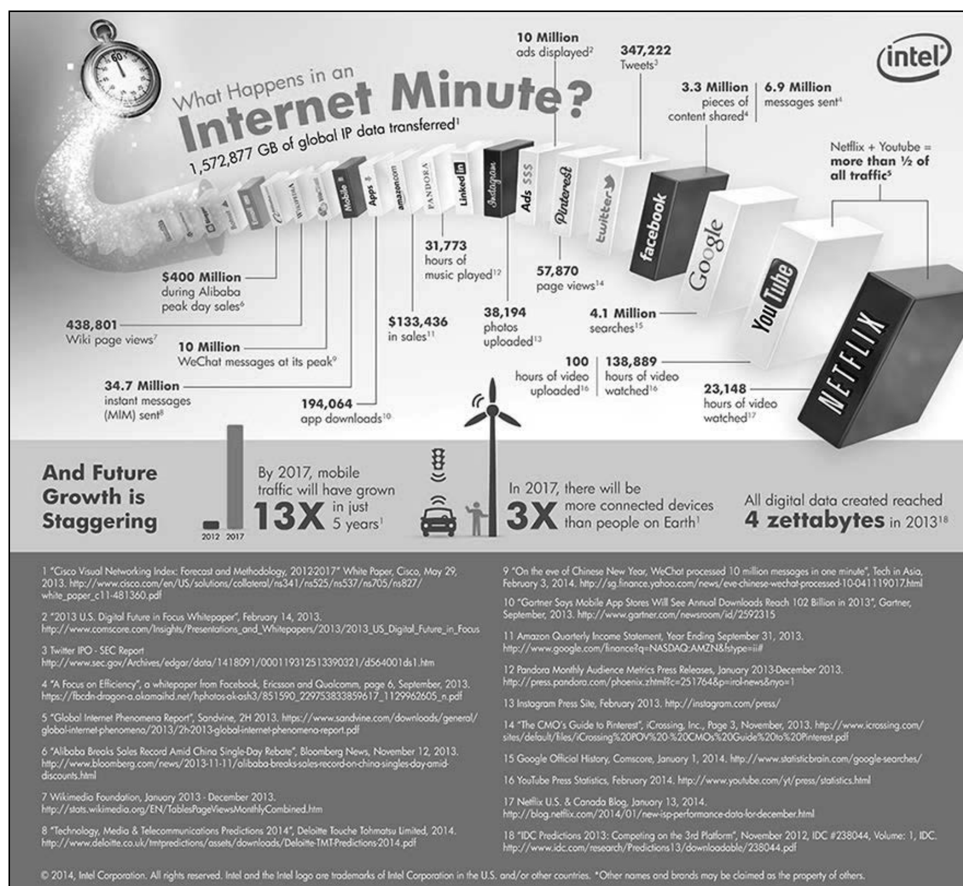



图1-1　通过不同互联网服务收集的海量数据

找到数据之后，我们面对的主要问题是要花时间与空间积累数据。这些数据只表明发生了什么，而不收集意图、动机或内在动机。最后，这样的数据中可能充斥着噪声，它们可能不完整、不一致，甚至会随时间发生变化。

获取数据的另一种方法是从各种传感器收集测量数据，比如移动设备中的惯性和位置传感器、环境传感器、软件代理关键性能监控指示器等。

## 1.3.2　生成数据

另一种替代方法是你自己生成数据，比如通过调查。做调查设计时，我们必须把注意力放在数据采集上，明确被调查的对象是谁。我们只能从那些愿意参与调查并回答问题的被调查者处获取数据，并且，被调查者可以做出符合他们自身形象与研究者期望的回答。

其次，我们也可以通过模拟收集数据。这个过程中，领域专家在微观水平上指定用户行为模型。例如，群体模拟（crowd simulation）需要指定不同类型的用户在群体中如何行动，比如是随大流还是伺机逃离等。接着在不同条件下运行模拟，观察不同条件下会发生什么（Tsai等，2011）。模拟适用于研究宏观现象与突现行为（emergent behavior），然而这些模拟往往很难在实际生活中进行验证。

此外，你可以设计一些实验，彻底覆盖所有可能的结果，让所有变量保持不变，一次只操作一个变量。使用这种方法的代价很高，但通常都能得到质量最好的数据。

### 1.3.3　采样陷阱

收集数据时可能碰到许多陷阱。为了证实这一点，我们先设想下面一个场景。假设存在一个通用的、不成文的规则，用于在学生之间免费发送普通邮件。如果你在贴邮票的地方写上“学生到学生”，这封邮件就会被免费投递到接收方。现在，假设雅各布给艾玛发送了一套明信片，并且假设艾玛真的收到了一些明信片。于是她就断定所有的明信片都寄到了，并且认为规则是真实有效的。也就是说，艾玛基于自己收到明信片这一事实就推断所有明信片都被寄到，然而她并没有那些雅各布寄出但未投递到的明信片信息。因此，艾玛无法在推断时把它们考虑进去。艾玛经历的就是“幸存者偏差”，即她只依据“幸存”的数据得出结论。需要指出的是，使用“学生到学生”邮票邮寄的明信片上都印有带圆圈的字母T，这表示“欠邮资”，邮资应该由接收方支付，包括小额罚金。然而，邮政公司常常要付出更多代价收取这些费用，因此一般不会这样做（Magalhaes，2010）。

另一个例子来自一项研究，这项研究发现平均死亡年龄最低的职业是“学生”。做学生不会造成你在年纪很小的时候死亡，只能说明你还年轻。这才是学生这个群体平均死亡年龄低的原因（Gelman与Nolan，2002）。

此外，一项研究发现，发生事故的司机中，只有1.5%报告说他们当时正在使用手机，而有10.9%的司机报告说是车中另一名乘客分散了他们的注意力。据此，我们能推断说，司机驾车时使用手机比与另一个乘客交谈更安全吗？（Uts，2003）为了回答这个问题，我们需要知道手机使用的盛行率。当时收集数据期间，相比于开车时使用手机，司机更有可能与车中的另一个人进行交谈。

## 1.4　数据预处理

数据预处理的目标是，用最可行的方式为机器学习算法准备数据，因为并非所有算法都可以用于处理缺少数据、额外属性以及非标准准值。

### 1.4.1　数据清洗

数据清洗也叫数据整理、数据清理，处理过程如下：

- 识别不准确、不完整、不相关、已损坏的数据，并在进一步处理之前移除；
- 分析数据，提取感兴趣的信息，或者验证数据格式是否合法；
- 将数据转换为常见的编码格式，比如utf-8、int32、时标或者标准范围；
- 将数据转换为常见数据模式，比如如果收集的温度数据来自不同类型的传感器，那么可能需要将其变换为具有相同结构的数据。

接下来，看看更具体的数据预处理步骤。

### 1.4.2　填充缺失值

一般来说，对于缺失值，机器学习算法工作得不是很好。但有极少数例外，比如决策树、朴素贝叶斯分类器，以及一些基于规则的学习器等。了解一个值缺失的原因至关重要，这些原因可能是多方面的，比如随机误差、系统误差、传感器噪声等。一旦找到缺失原因，就可以采用多种方法处理缺失值，常见处理方法如下。

- **移除实例**：如果有足够多的数据，并且其中只有几个非相关实例有一些缺失值，那么移除这些实例是安全的。
- **移除属性**：当大部分值缺失、值为常量，或者当前属性与另一个属性有强相关关系时，移除该属性是有意义的。
- **指派为特殊值N/A**：有时缺失值是由正当理由引起的，比如值超出指定范围、离散属性值未定义、无法获取或测量得到的值——这个值也可能是指示器（indicator）。比如，一个人从来不评价电影，那么他对这部电影的评分就不存在。
- **填入属性平均值**：如果拥有的实例数量有限，那么不能移除实例或属性。这种情况下，我们可以对缺失值进行估算，比如把属性的平均值或相似实例的平均值作为缺失值进行填充。
- **依据其他属性值进行预测**：如果属性有时间依赖关系，那么可以根据之前的已有值预测缺失值。

如上所述，出现缺失值的原因多种多样，因此了解值缺失或损坏的原因非常重要。

### 1.4.3　剔除异常值

异常值是指数据中那些与其他数值相比有较大差异的数值，这些异常值对所有学习算法都有不同程度的影响。异常值可能是极端值，可以通过置信区域检测，并可借助阈值剔除。最好的方

法是先对数据进行可视化，然后检查可视化图形，从中找出异常值。图1-2的可视化图形是个例子。请注意，数据可视化仅适用于低维数据。

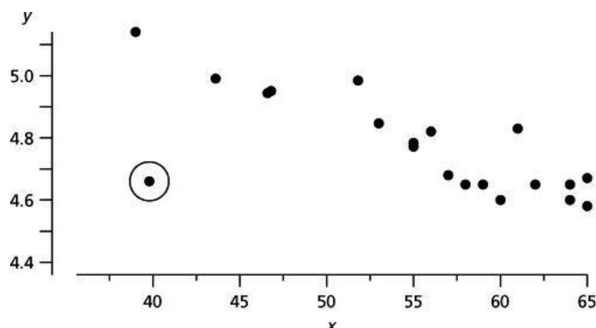



图1-2　可视化图形

## 1.4.4　数据转换

数据转换技术将数据集转换为机器学习算法要求的格式，用作机器学习算法的输入。数据转换甚至可以帮助算法学得更快，获得更好的性能，比如标准化。假设数据服从高斯分布，采用如下方式做值变换：均值为0，标准差为1。

$$X=\frac{X-\text{mean}(X)}{\text{st.dev}}$$

另一方面，归一化将属性值按比例缩放，使之落入一个小的特定区间，通常是[0, 1]。

$$X=\frac{X-\min}{\max-\min}$$

许多机器学习工具箱自动对数据做归一化与标准化处理。

最后一个变换技术是离散化，用于将一个连续特征的范围切分为若干区间。为什么要这样做？因为有些算法，比如决策树、朴素贝叶斯算法，更擅长处理离散属性。最常用的选取区间（离散化）的方法如下。

- **等宽离散化**：该方法将连续变量的值域划分成$k$个具有相同宽度的区间。
- **等频离散化**：假设有$N$个实例，$k$个区间中的每一个都包含大约$N/k$个实例。
- **最小熵（度量体系的混乱程度）离散化**：该方法会递归地分割区间，直到区间分割引起的熵减大于熵增（Fayyad和Irani，1993）。

其中，前面两个方法需要手工指定区间数量，而最后一种方法则自动设置区间数量。但后者需要分类变量，这意味着它不能用于无监督机器学习任务。

### 1.4.5　数据归约

数据归约用于处理大量属性与实例，属性数对应于数据集的维度数。具有较低预测能力的维度不仅对整个模型的贡献率非常小，还会带来许多危害。比如，一个拥有随机值的属性可能产生一些随机模式，这些随机模式会被机器学习算法识别。

为了解决这个问题，第一种处理方法是把这些属性剔除，换言之，只保留那些最看好的属性。这个过程称为特征选取或属性选取，具体可使用的方法有ReliefF、信息增益、基尼指数等，它们主要面向离散属性。

另一个处理方法是专注于连续属性，将数据集从原始维度转换到低维空间。例如，假设有一组三维空间中的点，我们可以将其映射到二维空间。这个过程中会丢失一些信息，但如果第三个维度是不相关的，则不会丢失很多信息，数据结构与相关性几乎都能完美保留。具体可用方法如下：

- **奇异值分解（SVD）**
- **主成分分析（PCA）**
- **神经网络自动编码器（Neural nets auto encoders）**

数据归约中的第二个问题是数据包含太多实例。这么多个实例可能是重复的，或者来自一个非常频繁的数据流。解决这个问题的方法是从中选用实例子集，选择时要保证所选数据的分布与原数据的分布相似，更重要的是观测的过程类似。减少实例数量的技术包括随机数据采样、数据分层法等。准备好数据后，接下来对数据进行分析与建模。

## 1.5　无监督学习

无监督学习是指，通过数据分析从没有标签的数据中发现隐藏的结构。由于数据不带有标签，所以我们无法通过误差测量对学过的模型做评价。即便如此，无监督学习仍然是个极其强大的工具。你是否曾经好奇过，亚马逊是如何预测你喜欢什么书的？在你还未做出选择时，Netflix如何知道你想看什么？这些问题的答案都可以在无监督学习中找到。下面给出一个类似的例子。

### 1.5.1　查找相似项目

许多问题都可以归结为查找元素相似集，比如购买了类似商品的顾客、包含相似内容的网页、含有相似对象的图像、访问过类似网站的用户等。

如果两个项目相距非常近，就可以将其视为是类似的。主要问题是如何表示每个项目，以及如何定义项目之间的距离。距离测量主要有两类：一类是欧氏距离（Euclidean distance），另一类是非欧距离（no-Euclidean distance）。

### 1. 欧氏距离

$n$维欧氏空间中，两个元素之间的距离基于元素在这个空间中的位置，常称为“p-范数距离”（p-norm distance）。常用的两个距离度量是L2与L1范数距离。

L2范数也叫欧氏距离，它是最常用的距离度量，用于度量二维空间中的两个元素相距多远。它是两个元素在每个维度上差的平方和的平方根，计算公式如下：

$$\mathrm{L_2norm}\ d(a,b)=\sqrt{\sum_{i=1}^{n}(a_i-b_i)^2}$$

L1范数又称曼哈顿距离（Manhattan Distance）、城市街区距离（City Block Distance）、出租车范数（Taxicab Norm），它是两个元素在每个维度上差的绝对值之和，计算公式如下：

$$\mathrm{L_1norm}\ d(a,b)=\sum_{i=1}^{n}\left|a_i-b_i\right|$$

### 2. 非欧距离

非欧距离基于元素的属性，而非它们的空间位置。其中较为著名的有杰卡德距离（Jaccard Distance）、余弦距离（Cosine Distance）、编辑距离（Edit Distance）、汉明距离（Hamming Distance）。

杰卡德距离计算两个集合间的距离。首先，计算两个集合的杰卡德相似系数（Jaccard Similarity），它被定义为“两个集合交集的元素个数除以并集的元素个数”，公式如下：

$$\mathrm{sim}(A,B)=\frac{A\cap B}{A\cup B}$$

杰卡德距离被定义为“1减去杰卡德相似系数”，公式如下：

$$d(A,B)=1-\mathrm{sim}(A,B)=1-\frac{A\cap B}{A\cup B}$$

余弦距离也称为余弦相似度（Cosine Similarity）。两个向量的余弦距离关注的是它们的方向，而非大小。因此，如果两个向量方向一致，那么它们的余弦相似度为1，而两个垂直向量的余弦相似度为0。假设有两个多维点，所对应的向量分别从原点(0, 0, 0, ..., 0)指向它们所在的位置。两个向量之间形成一个夹角，它们的余弦距离就是向量的标准点积，如下：

$$d(A,B)=\mathrm{arcos}\frac{AxB}{\|A\|\|B\|}$$

余弦距离通常用于高维特征空间，比如文本挖掘。一个文本文档代表一个实例，特征对应于不同单词，它们的值对应于单词在文档中出现的次数。通过计算余弦相似度，我们能够了解两个文档内容的相似程度。

编辑距离用于比较两个字符串。两个字符串a（$=a_1a_2a_3...a_n$）与b（$=b_1b_2b_3...b_n$）之间的距离是指，从字符串a转成字符串b所需的最少编辑操作数，允许的编辑操作包括插入一个字符、删除一个字符。比如，有两个字符串a=abcd和b=abbd，将字符串a转成b时，必须先删除字符串b中的第三个字母b，然后在删除位置插入字母c。整个过程中没有比这更少的操作数了，所以字符串a到b的编辑距离就是$d$(a, b)=2。

汉明距离用于比较两个大小相同的向量，计算它们不同的维数。换言之，该距离指的是从一个向量变换为另一个向量所需的替换数。

许多距离可以用于度量各种属性，比如相关性度量两个元素之间的线性关系，马氏距离（Mahalanobis Distance）度量一个点与其他点所服从的分布之间的距离，SimRank基于图形理论衡量元素呈现结构的相似程度，等等。正如你想的那样，为你的问题选择并设计合适的相似性度量后，就完成了一大半工作。A.A. Goshtasby所著的*Image Registration: Principles, Tools and Methods*一书的第2章中，关于相似性度量评估的讲解令人印象深刻，可以参考学习。

3. 维数灾难

维数灾难是指我们拥有大量特征——通常有成千上百个——时，这些特征会产生一个带有稀疏数据的极大空间，以致距离异常。比如在高维空间中，几乎所有点对彼此之间都是等距的。其实，几乎所有点对的距离都接近平均距离。维数灾难的另一个表现是，任意两个向量几乎都是垂直的，这意味着所有夹角都接近90°。实际上，这让任何距离度量都失去了意义。

使用某种减少特征数量的数据归约技术或许可以解决维数灾难问题。比如，可以运行ReliefF等特征选择算法或PCA等特征提取/缩减算法减少特征数量。

## 1.5.2　聚类

聚类技术根据某种距离度量，将类似的实例归入相应的簇。主要思想是将类似（相互靠得很近）的实例放入同一个簇，同时让不相似（彼此离得很远）的点位于不同的簇。图1-3展现了簇的样子。

聚类算法有两个最基本的方法。第一个是**分层**（**hierarchical**）或**凝聚**（**agglomerative**）方法，先将每个点作为它自己的簇，然后不断把最相似的簇合并在一起。合并达到预先指定的簇数时，或者待合并的簇覆盖一大片区域时，就停止合并操作。

另一个方法基于点分配（Point Assignment）。首先估计（比如随机）初始的簇中心（即簇质心），然后将每个点分配到离它最近的簇，直到分配完所有点。最有名的算法是**K均值聚类算法**。

K均值聚类算法中，把那些相互间尽可能远的点选为初始的簇中心，或者（分层）聚集数据样本并选取离每个簇（共$k$个簇）中心最近的点作为初始的簇中心。

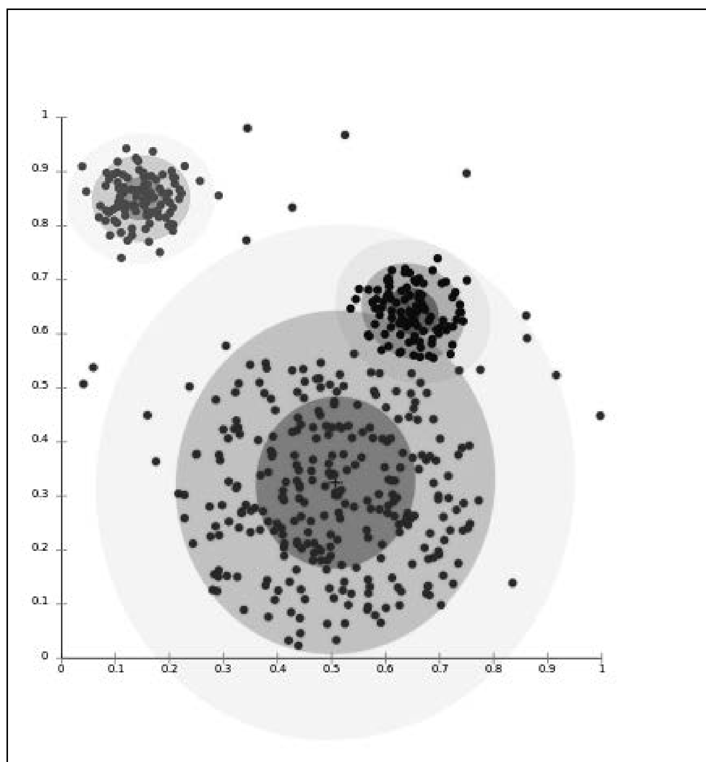



图1-3 簇

## 1.6 监督学习

许多令人惊叹的技术背后隐藏着一个关键概念——监督学习，这些技术包括声音识别、垃圾邮件过滤、图像中的面部识别、侦测信用卡欺诈等。更正式地讲，给定一组学习样本D，用特征$X$进行描述，监督学习的目标是找到一个函数对目标变量$Y$进行预测。函数$f$描述特征$X$与类$Y$之间的联系，称为模型：

$$f(X) \to Y$$

监督学习算法的通用结构由如下决策定义（Hand等，2001）：

- 定义任务；
- 确定机器学习算法，它会产生特定归纳偏置，即先验假设，这是针对目标概念做出的；
- 确定得分函数与代价函数，比如信息增益、均方根误差等；
- 确定最优/搜索方法优化得分函数；
- 找到一个函数，用以描述$X$与$Y$之间的关系。

上面许多决定已经由任务的类型与我们拥有的数据集确定了。接下来将详细学习分类与回归方法，以及相应得分函数的有关内容。

## 1.6.1　分类

分类可以处理离散类，其目标是对目标变量中的互斥值之一进行预测。一个应用例子是做信用评估，最终预测结果是判断目标人物的信用是否可靠。最流行的算法有决策树、朴素贝叶斯分类器、支持向量机、神经网络以及集成算法（Ensemble Method）。

### 1. 决策树学习

决策树学习过程中会创建一个分类树，树的每一个节点对应一个属性，边对应属性一个可能的值（或区间），节点由此而生，每个叶子对应一个类标签。决策树可以可视化并以明确的方式表示预测模型，这让它成为一个很透明（白箱）的分类器。比较有名的算法有ID3与C4.5，此外还有许多可选实现与改进算法（比如Weka中的J48）。

### 2. 概率分类器

给定一组属性值，概率分类器（Probabilistic Classifiers）可以对一组类的分布进行预测，而不预测一个确切的类。这可以作为确定性程度来使用，即分类器对自己的预测有多大把握。最基本的分类器是朴素贝叶斯分类器，它也是最理想的分类器——当且仅当属性是条件独立的。但不幸的是，这在实际情况中极其少见。

其实有一个称为概率图模型的庞大分支，包括成百上千的算法，比如贝叶斯网络、动态贝叶斯网络、隐马尔可夫模型、条件随机场（不仅可以处理两个属性间的特定关系，还可以处理现时依赖性）。关于这个主题，Karkera写了一本很棒的入门书*Building Probabilistic Graphical Models with Python*，Koller与Friedman出版了一本详尽的理论“圣经”——《概率图模型：原理与技术》。

### 3. 核方法

通过对模型应用核技巧（kernel trick），用核函数（kernel function）替代模型的特征（预测器），可以将任意一个线性模型转换为非线性模型。换言之，核技巧隐式地将数据集变换成更高维度。核技巧充分利用了“分离更高维的实例通常更容易”这一事实。可以使用核技巧的算法包括核感知器、**支持向量机（SVM）**、高斯过程、PCA、典型相关分析、岭回归、谱聚类、线性自适应过滤器等。

### 4. 人工神经网络

人工神经网络是受生物神经网络结构的启发而提出的，可以用于机器学习，也可以进行模式识别。人工神经网络通常解决回归与分类问题，包含各种算法以及各种问题类型的变种。比较流

行的分类方法有**感知器**、**受限玻尔兹曼机**、**深度信念网络**（**Deep Belief Network**）。

### 5. 集成学习

集成方法由一系列不同的弱模型组成，以此获得更好的预测能力。先单独训练各个模型，然后采用某种方式将其预测组合起来，以产生更全面的预测。因此，集成体包含针对数据的多种建模方式，希望能够产生更好的结果。集成学习是机器学习算法中非常强大的工具，也很流行，包括Boosting方法、Bagging方法、AdaBoost、随机森林。这些方法的主要不同在于它们组合的学习器的类型以及选用组合方式。

### 6. 分类评估

我们的分类器工作效果很好吗？这个分类器比另一个要好吗？在分类中，我们计算分对与分错的次数。假设有两个可用的分类标签——yes与no，有4种可能的结果，如表1-3所示。

- 真正–命中：这表示一个yes实例被正确地预测为yes。
- 真负–正确否定：这表示一个no实例被正确地预测为no。
- 假正–误警：这表示一个no实例被预测为yes。
- 假负–未命中：这表示一个yes实例被预测为no。

**表1-3 两个分类标签的可能结果**

| | | 预测为正？ | |
|---|---|---|---|
| | | Yes | No |
| 真正吗？ | Yes | TP-真正 | FN-假正 |
| | No | FP-假正 | TN-真负 |

对分类器的性能进行度量的两个最基本度量值是分类错误与分类精度，如下：

$$分类错误=\frac{错误数}{总数}=\frac{FP+FN}{FP+FN+TP+FN}$$

$$分类精度=1-错误=\frac{正确数}{总数}=\frac{TP+TN}{FP+FN+TP+FN}$$

这两个度量值的主要问题是，它们不能处理不平衡类。对一笔信用卡交易是否为欺诈进行分类就是不平衡类问题的一个例子：正常交易占99.99%，欺诈仅占极小数。对于每笔交易，分类器将其判断为正常交易，这种准确率可达99.99%，但我们主要感兴趣的是那些极少出现的几个分类。

(1) 准确率与召回率

这个解决方案用到了两个不包含TN（正确否定）的度量值，它们定义如下。

- 准确率：被分类器判定为正的所有样本实例（TP+FP）中，被正确判断为正（TP）的正例样本所占比重。

$$准确率=\frac{TP}{TP+FP}$$

- 召回率：在总正例样本（TP+FN）中，被正确判定为正（TP）的正例所占比重。

$$召回率=\frac{TP}{TP+FN}$$

常见的做法是，把两个度量值组合起来，形成*F*值（F-measure）作为加权平均值；计算分数的同时考虑准确率和召回率，分数的最好值为1，最差值为0，计算公式如下：

$$F\ 值=\frac{2*准确率*召回率}{准确率+召回率}$$

(2) Roc曲线

大多数分类算法都会返回一个分类置信度，记作$f(X)$，它反过来计算预测。沿用前面信用卡欺诈的例子，规则可能如下：

$$F(x)=\begin{cases}欺诈，如果 f(X)>阈值\\ 非欺诈，其他\end{cases};$$

阈值决定错误率与真正率。我们可以把所有可能的阈值结果绘制成**ROC曲线**（**受试者工作特征曲线**），如图1-4所示。

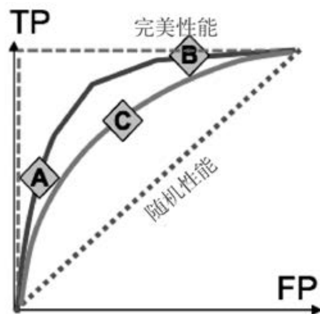



图1-4　阈值的ROC曲线

图中短点虚线表示的是随机预测器，长点虚线表示的是完美预测器。为了判断分类器A是否优于分类器C，我们对曲线以下的区域进行比较。

大多数工具箱都提供上面提到的所有度量指标，且开箱即用。

## 1.6.2　回归

回归方法处理连续的目标变量，这与使用离散目标变量的分类方法不同。例如，预测未来几

天的室外温度时，我们会使用回归方法，而分类方法只能预测未来几天是否下雨。一般来说，回归过程评估的是各种特征之间的联系，即特征变化是如何改变目标变量的。

### 1. 线性回归

最基本的回归模型假定特征与目标变量之间有线性依赖关系。这个模型经常使用最小二乘法进行拟合，它是使误差的平方最小的模型。许多情况下，线性回归不能对复杂关系进行建模。比如，图1-5显示了4组不同的点，它们有相同的线性回归曲线。其中左上模型描述了数据的总体趋势，可以认为模型是合适的；左下模型对数据点的拟合更好，但有一个离群点（你应该小心检查它）；右上与右下的线性模型完全偏离了下层的数据结构，不能将其视为合适的模型。

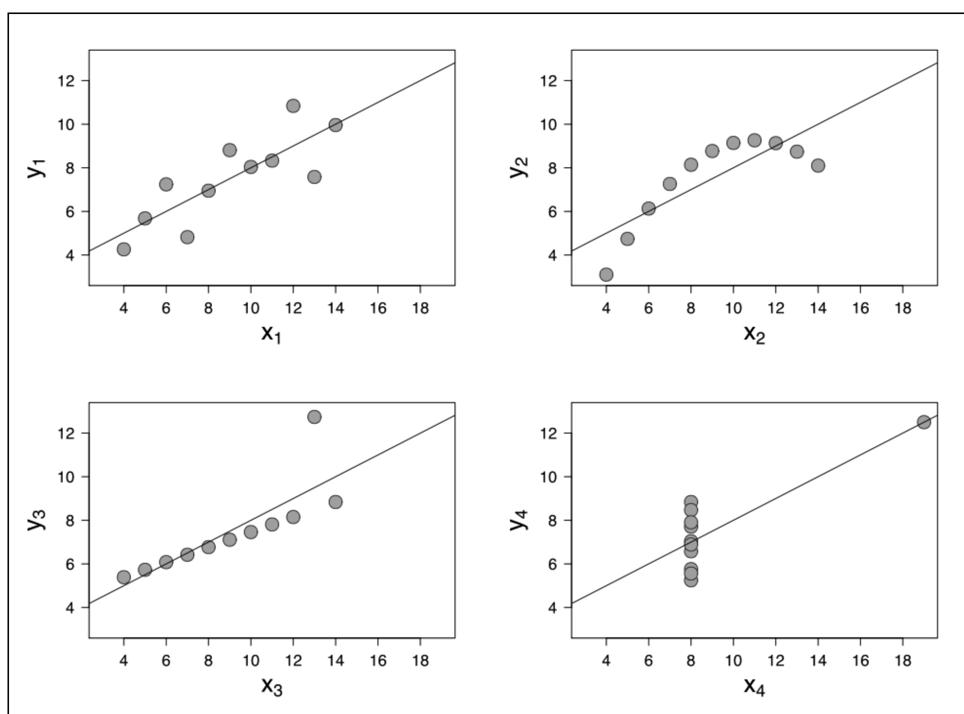



图1-5 线性回归模型

### 2. 回归评估

回归中，我们从输入*X*预测数值*Y*，这些预测通常是错误的、不准确的。我们要问的主要问题是：这些预测值与实际值相差多少？换言之，我们要测量预测值与实际值之间的距离。

(1) 均方误差

均方误差（mean squared error）是预测值与实际值差的平方和的平均值，计算公式如下：

$$\text{MSE}(X,Y)=\sqrt{\frac{1}{n}\sum_{i=1}^{n}(f(X_i)-Y_i)^2}$$

均方误差对异常值非常敏感。比如，99个准确率100%的预测加上1个准确率为90%的预测，得分和100个准确率为99%的预测一样。而且，均方误差对平均值也敏感。因此，人们更多使用相对平方误差（relative squared error）将预测器的MSE与均值预测器——总是用于预测平均值——的MSE进行比较。

(2) 平均绝对误差

平均绝对误差（mean absolute error）是预测值与实际值差的绝对值之和的平均值，计算公式如下：

$$\text{MAS}(X,Y)=\frac{1}{n}\sum_{i=1}^{n}\left|f(X_i)-Y_i\right|$$

MAS对异常值不太敏感，但它对平均值与规模是敏感的。

(3) 相关系数

相关系数以两变量与各自平均值的离差为基础，通过两个离差相乘反映两变量之间相关程度。若相关系数为负值，则表示弱相关；若为正值，则表示强相关；若为0，则表示不相关。实际值$X$与预测值$Y$之间的相关系数定义如下：

$$\text{CC}_{XY}=\frac{\sum_{i=1}^{n}(X_i-\bar{X})(Y_i-\bar{Y})}{\sqrt{\sum_{i=1}^{n}(X_i-\bar{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i-\bar{Y})^2}}$$

相关系数对平均值与规模完全不敏感，对异常值不太敏感。它能获取相对排序（relative ordering），对文档关联性与基因表达等任务进行分级排列时很有用。

## 1.7　泛化与评估

模型创建好之后，我们如何知道它针对新数据正常工作？这个模型有什么好？为了回答这些问题，首先学习模型泛化（model generalization），然后了解如何对模型在新数据上的性能进行评估。

### 欠拟合与过拟合

对预测器训练得到的模型可能太复杂或者太简单。低复杂度模型（最左边模型）可能像预测最频繁或平均类值一样简单，而高复杂度模型（最右边模型）能够表示训练实例。模型太刚性（如

图1-6左侧模型）就不能获得复杂模式，而模型太柔性（如图中右侧模型）就会把训练数据中的噪声也一起融合进去。我们面临的主要挑战在于选择合适的学习方法及其参数，只有这样，经过学习得到的模型才能在新数据上有良好的表现（见图1-6中间模型）。

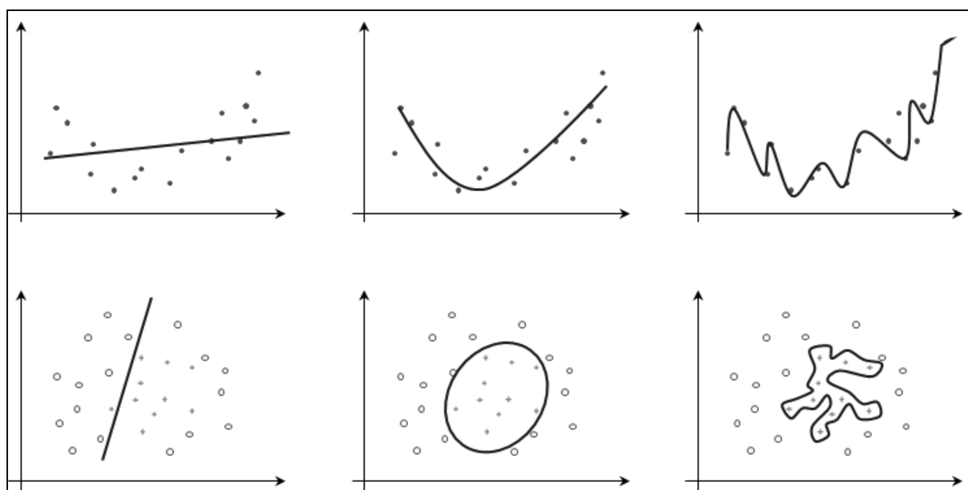


图1-6　不同类型的模型

图1-7反映了训练集中的错误是如何随着模型复杂度减少的。简单刚性模型对数据的拟合度不够，导致大量错误。随着模型复杂度的增加，它可以更好地描述训练数据的结构，错误必然会减少。如果模型太复杂，极有可能对训练数据拟合过度，预测错误随之增加。

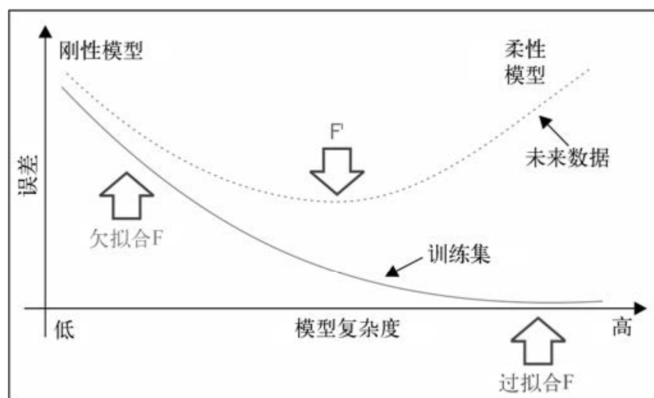



图1-7　训练集中的错误随着模型复杂度的增加而减少

根据任务复杂度与数据可用性，我们可以把分类器的结构调整得更简单或者更复杂。大部分学习算法支持用户做如下调整。

- **回归**：多项式的次数
- **朴素贝叶斯**：属性数目
- **决策树**：树节点的数目、修剪置信度
- **k最近邻算法**：邻居数目，基于距离的邻居权重
- **SVM**：核类型，代价参数
- **神经网络**：神经元与隐藏层的数目

通过调整，我们希望把泛化误差（Generalization error）最小化，即让分类器在未来数据上有良好的表现。遗憾的是，我们无法计算真正的泛化误差，但可以做估计。然而，如果模型在训练数据上表现良好，但在测试数据上表现较差，说明这个模型极有可能过拟合。

### 1. 训练集与测试集

为了估算泛化误差，我们将拥有的数据划分成两部分：训练数据与测试数据。根据一般经验，划分数据时，训练数据与测试数据的比例应该是70:30。首先使用训练数据对预测器进行训练，然后使用预测器对测试数据进行预测，最后计算误差（预测值与真实值之差）。这样就可以对真实的泛化误差进行评估了。

评估基于如下两个假设：首先，假设测试集是来自数据集的无偏样本集；其次，假设新数据将根据训练与测试样本进行分布重组。第一种假设可以通过交叉验证与层化处理（Stratification）的方法得到实现。此外，虽然这种情况很少见，但如果真的没有足够多的数据用作单独的测试集，那么学习算法就会运行得很不理想——因为它们接收不到足够多的数据。这种情形下，可以使用交叉验证。

### 2. 交叉验证

交叉验证将数据集划分成*k*个子集，它们大小差不多，比如图1-8显示的5个子集。首先，将2~5子集用于学习，将1子集用于训练。然后重复这个过程5次，每次多拿出一个集合用于测试，并计算5次重复的平均误差。

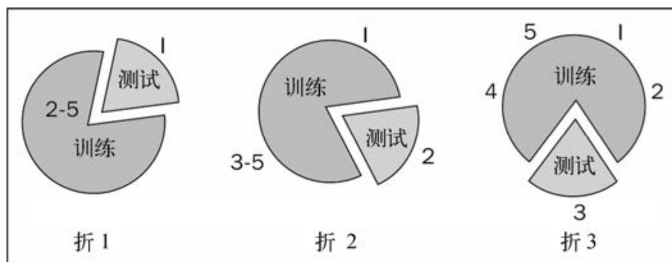



图1-8　交叉验证

借助这种方法，可以将所有数据用于学习与测试，但要避免使用相同数据训练与测试模型。

#### 3. 留一法交叉验证

留一法交叉验证是交叉验证的一个极端例子。留一法交叉验证中，折（fold）的数目等同于实例的数目。先留出一个实例，其余实例全部用来学习，然后用留出的实例测试模型。针对所有实例重复上述过程，这样每个实例只有一次用来做验证。我们拥有的学习样本有限时——比如少于50个——建议使用该方法。

#### 4. 层化处理

层化处理方法用于选取样本子集，选择时，每一层（fold）都大致包含相同比例的类值。如果一个类是连续的，那么这些分层都会被选择，这样平均响应值在所有层差不多都一样。层化处理可以和交叉验证、单独训练集或测试集一起使用。

## 1.8　小结

本章复习了机器学习的基础知识，重温了机器学习的应用流程，还阐述了机器学习的主要任务、方法与算法。下一章将介绍有哪些可用的Java机器学习库，以及它们可以用来做什么任务。

第 2 章

# 面向机器学习的Java库与平台

自己动手实现机器学习算法可能是最好的学习方法，但如果你站在巨人肩上，充分利用现有开源库，就能更快地推进自己的项目。

本章将学习Java语言中与机器学习相关的各种库与平台，了解每个库的功能，以及可以用它们解决的问题。

本章将学习如下内容。

- 实现机器学习应用时需要具备的Java环境
- Weka：一个通用的机器学习平台
- Java机器学习库：一系列机器学习算法
- Apache Mahout：一个可伸缩的机器学习平台
- Apache Spark：一个分布式机器学习库
- Deeplearning4j：一个深度学习库
- MALLET：一个文本挖掘库

此外，我们还将讨论如何使用这些库与其他组件为单机与大数据应用创建完整的机器学习应用栈。

## 2.1 Java 环境

新机器学习算法常常先出现在大学的实验室中，通常组合了几种语言，比如shell脚本、Python、R、MATLAB Java、Scala、C++，用于证明一个新概念，并对属性做理论分析。一个算法可能要经过漫长的重构之路，才能进入一个带有标准输入/输出与用户界面的库。Python、R、MATLAB相当流行，它们主要用来编写脚本、做研究与实验。另一方面，Java是事实上的企业级语言，这归因于它支持静态类型，拥有健壮的IDE支持，以及良好的可维护性、不错的线程模型、高性能并发数据结构库。此外，还有许多可用的Java机器学习库，用户可以很方便地将其应用于

现有的Java应用程序，充分利用机器学习的强大能力。

## 2.2　机器学习库



基于Java的开源机器学习项目超过70个，你可以在MLOSS.org网站中看到这些项目的列表。除此之外，可能还有很多未列出的项目存在于各大学的服务器、GitHub或BitBucket上。我们将学习几个主要的库与平台，了解它们能够解决哪类问题、支持哪些算法，以及使用哪种数据。

### 2.2.1　Weka

Weka是**Waikato Environment for Knowledge Analysis**（**怀卡托智能分析环境**）的缩写，它是由新西兰怀卡托大学开发，可能是最著名的Java机器学习库。Weka是一个通用库，能够完成各种机器学习任务，比如分类、回归、聚类，其特色是拥有丰富的图形用户界面、命令行界面与Java API。更多内容请参考http://www.cs.waikato.ac.nz/ml/weka/。

写作本书之时，Weka总共包含267种算法：数据预处理（82）、特征选择（33）、分类与回归（133）、聚类（12）、关联规则挖掘（7）。图形界面适用于数据探索，而Java API允许开发新的机器学习方案，并将相关算法应用到你的应用程序。




图2-1　Weka

Weka在**GNU通用公共许可协议**下发行，这意味着你可以复制、发行与修改。但要在源文件中跟踪变化，并且遵守GNU GPL协议。你甚至可以商业方式发行，但必须开放源代码或者获得商业许可证。

Weka除了支持几种常见的文件格式之外，还有自己默认使用的专有数据格式——ARFF，这种数据格式使用属性数据对（attribute-data pairs）描述数据。ARFF文件主要包含两部分，第一部分是头部定义，指定所有属性（即特征）和对应的类型，比如分类型（nominal）、数值型（numeric）、日期型（date）、字符串型（string）；第二部分是包含数据的数据区，其中每一行对应一个实例。头部定义中的最后一个属性被隐式地看作目标变量，缺失数据用问号表示。回想一下第1章举过的例子，采用ARFF文件格式书写Bob实例，如下所示：

```
@RELATION person_dataset

@ATTRIBUTE `Name`  STRING
@ATTRIBUTE `Height`  NUMERIC
```

```
@ATTRIBUTE `Eye color`{blue, brown, green}
@ATTRIBUTE `Hobbies`  STRING

@DATA
'Bob', 185.0, blue, 'climbing, sky diving'
'Anna', 163.0, brown, 'reading'
'Jane', 168.0, ?, ?
```

如上所示，整个文件由3个区块组成，第一区块由`@RELATION <String>`关键字开始，指定数据集名称；第二区块以`@ATTRIBUTE <String>`关键字开始，后面跟着属性名和相应的类型，可用的类型有`STRING`、`NUMERIC`、`DATE`，以及一组分类值——最后一个属性被隐式假定为我们要预测的目标变量；最后一个区块以`@DATA`关键字开始，后面的每一行就是一个实例。实例的各个属性值以逗号分隔，排列顺序与第二个区块中的属性排列顺序一致。



在第3章与第4章中，你将看到更多Weka的应用示例。

想学习更多关于Weka的内容，你可以阅读快速入门书*Weka How-to*并开始编码，或者阅读《数据挖掘：实用机器学习工具与技术》深入了解相关理论背景。

Weka的Java API组织在如下顶层包中。

- `weka.associations`：包含数据结构与关联规则学习算法，如**Apriori**、**Predictive Apriori**、**FilteredAssociator**、**FP-Growth**、**Generalized Sequential Patterns (GSP)**、**Hotspot**、**Tertius**。
- `weka.classifiers`：包含监督学习算法、评估器（evaluators）与数据结构。该包进一步划分为如下组件。
  - `weka.classifiers.bayes`：该组件实现了贝叶斯方法、包括朴素贝叶斯、贝叶斯网络、贝叶斯逻辑回归等。
  - `weka.classifiers.evaluation`：包含用于分类预测与数值预测的监督评估算法，比如评估统计、混淆矩阵、ROC曲线等。
  - `weka.classifiers.functions`：包含回归算法，比如线性回归、保序回归、高斯过程、支持向量机、多层感知器、表决感知器及其他。
  - `weka.classifiers.lazy`：包含基于实例的算法，比如k最近邻算法、K*、懒惰式贝叶斯规则。
  - `weka.classifiers.meta`：包含监督学习元算法，比如AdaBoost、bagging、Additive Regression、RondomCommittee等。
  - `weka.classifiers.mi`：包含多实例学习算法，比如Citation k-nn、多样密度算法、MI AdaBoost及其他。
  - `weka.classifiers.rules`：包含决策表和基于separate-and-conquer方法、Ripper、Part、Prism等的决策规则。

- `weka.classifiers.trees`：包含各种决策树算法，比如ID3、C4.5、M5、功能树、逻辑树、随机森林等。

- `weka.clusterers`：包含聚集算法，比如K均值、Clope、Cobweb、DBSCAN层次聚类等。
- `weka.core`：包含各种实用工具类、数据展示、配置文件等。
- `weka.datagenerators`：包含针对分类、回归与聚集算法的数据生成器。
- `weka.estimators`：包含针对离散/名义域（discrete/nominal domains）、条件概率估计等各种数据分布估测器（data distribution estimators）。
- `weka.experiment`：包含一组支持试验运行的类，包括必需的配置、数据集、模型设置与统计。
- `weka.filters`：包含基于属性与基于实例的选择算法，为监督学习与非监督学习做数据处理。
- `weka.gui`：包含实现了**浏览器**、**实验器**（**experimenter**）、**知识流**（**knowledge flow**）程序的图形界面。浏览器允许查看数据集、算法与相应参数、带有散点图的可视化数据集以及其他可视化界面。实验器用来设计批量实验，但只能用在分类与回归问题上。知识流实现了一个可视化的拖放用户界面，用来创建数据流，比如装载数据、应用过滤器、创建分类器以及做评估。

### 2.2.2 Java 机器学习

Java机器学习库——简称Java-ML——由一系列机器学习算法组成，对同种类型的算法提供通用接口。因此，它唯一的特色是Java API，主要用户是软件工程师与程序员。Java-ML包含的算法涉及数据处理、特征选取、分类与聚集，它的另外一个特色是提供了几个Weka桥，方便用户直接通过Java-ML API访问Weka算法。你可以从http://java-ml.sourceforge.net下载，写作本书之时，它的最新版本是2012年发布的。

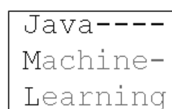



图2-2 Java-ML

Java-ML还是一个通用的机器学习库。相比于Weka，它提供了更一致的接口，以及最新算法的实现——这在其他包中是没有的，比如大量最新的相似性度量法与特征选取技术、动态时间规整算法、随机森林属性计算等。Java-ML也遵循GNU GPL许可协议。

Java-ML几乎可以支持所有文件类型，但要求文件中的每一行就是一个数据样本，各个特征通过逗号、分号、制表符进行分隔。

Java-ML库围绕如下顶层包进行组织。

- `net.sf.javaml.classification`：包含各种分类算法，比如朴素贝叶斯算法、随机森林、bagging、自组织映射、k最近邻算法等。
- `net.sf.javaml.clustering`：包含聚类算法，比如K均值、自组织映射、空间聚类、Cobweb、AQBC以及其他。
- `net.sf.javaml.core`：包含表示实例与数据集的类。
- `net.sf.javaml.distance`：包含测量实例距离与相似度的算法，比如**切比雪夫距离**、余弦距离/相似度、欧氏距离、杰卡德距离/相似度、**马氏距离**、**曼哈顿距离**、**闵可夫斯基距离**、**皮尔逊相关系数**、**Spearman's footrule**距离、**动态时间规整**（**DTW**）等。
- `net.sf.javaml.featureselection`：包含用于特征评估、评分、选择、排名的算法，比如信息增益率、ReliefF、KL散度、对称不确定性（symmetrical uncertainty）等。
- `net.sf.javaml.filter`：包含用于操作实例的方法，所允许的操作有过滤、移除属性、设置类或属性值等。
- `net.sf.javaml.matrix`：实现了内存或基于文件的数组。
- `net.sf.javaml.sampling`：实现了采样算法，用于选取数据集子集。
- `net.sf.javaml.tools`：包含各种实用工具方法，涉及数据集、实例操作、序列化、Weka API接口等。
- `net.sf.javaml.utils`：包含与算法有关的实用工具方法，比如统计、数学方法、列联表以及其他。

### 2.2.3 Apache Mahout

Apache Mahout项目的目标是创建一个可伸缩的机器学习库。它建立在可扩展的、分布式架构之上——比如Hadoop，使用MapReduce范式，支持在服务器集群上使用并行、分布式算法处理与生成大型数据集。




图2-3 Mahout

Mahout的特色是针对可扩展算法（这些算法用来做聚类、分类与协同过滤）提供了控制台接口与Java API。它可以解决三种商业问题：商品推荐（为人们推荐商品，比如“喜欢这部电影的客户还喜欢……”）；聚类（比如把某个文本文档归入相关文档组）；分类（比如学习将哪个主题指派给未打标签的文档）。

Mahout的发行遵守“商业友好”Apache许可证，这意味着你可以自由使用，但必须将Apache许可证包含并显示到你的程序版权声明中。

Mahout专有库如下所示。

- `org.apache.mahout.cf.taste`：包含协同过滤算法，其中有基于用户的，有基于商品的，还有带有ALS的矩阵分解法。
- `org.apache.mahout.classifier`：包含内存中与分布式的实现，涉及逻辑回归、朴素贝叶斯、随机森林、**隐马尔可夫模型**与多层感知器。
- `org.apache.mahout.clustering`：包含聚类算法，比如Canopy聚类、K均值、模糊K均值、Streaming K均值与谱聚类。
- `org.apache.mahout.common`：包含算法的实用工具方法，比如距离、MapReduce操作、迭代器等。
- `org.apache.mahout.driver`：实现了一个通用驱动程序，用来运行其他类的主方法。
- `org.apache.mahout.ep`：使用记步变异（recorded-step mutation）实现的进化优化算法。
- `org.apache.mahout.math`：包含各种数学实用方法与在Hadoop中的实现。
- `org.apache.mahout.vectorizer`：包含用于数据表示、操作与执行MapReduce任务的类。

### 2.2.4 Apache Spark

Apache Spark也可以简称为Spark，它是建立在Hadoop之上的大型数据处理平台。与Mahout相比，它并没有绑定到MapReduce范式之上。相反，它使用内存缓存提取工作数据集，处理数据并重复查询。有报道称，与直接工作在磁盘数据之上的Mahout实现相比，Spark的处理速度最高可快10倍。可以从https://spark.apache.org下载。




图2-4 Apache Spark

很多框架、库建立在Spark之上，比如用于处理图形的**GraphX**、处理实时数据流的**Spark Streaming**，以及进行机器学习（分类、回归、协同过滤、聚类、降维、优化）的MLlib库。

Spark的MLlib可以使用基于Hadoop的数据源，比如**Hadoop分布式文件系统**（**HDFS**）、HBase以及本地文件。支持的数据类型如下。

- **本地向量**：它被存储在一台单独的机器上，分为稠密向量与稀疏向量两种。前者表现为一个双精度类型值的数组，比如(2.0, 0.0, 1.0, 0.0)；而后者表现为向量长度、索引数组与

双精度浮点型值的数组，比如[4, (0, 2), (2.0, 1.0)]。

- **标注点（Labeled point）**：它用在监督学习算法中，由带有双精度类值的本地向量组成。标签可以是类索引、二分类结果或多个类索引列表（多类分类）。比如，一个带标记的稠密向量可以表示为[1.0, (2.0, 0.0, 1.0, 0.0)]。
- **本地矩阵（Local matrix）**：它在单机上存储稠密矩阵，由矩阵维数与一个双数组（以列为主序排列）定义。
- **分布式矩阵（Distributed matrix）**：用于操作存储在Spark **RDD**（**弹性分布式数据集**）中的数据，表示一系列可以并行操作的元素。有三种表示：行矩阵（row matrix），每一行都是一个可存储在单机上的本地向量，行索引无意义；带索引的行矩阵（indexed row matrix），类似于行矩阵，但是行索引有意义，即能对行进行标识与连接；坐标行矩阵（coordinate matrix），当行无法存储在单机上并且矩阵非常稀疏时使用。

Spark的MLlib API库针对各种学习算法提供了接口与实用工具，大致如下。

- `org.apache.spark.mllib.classification`：包含二元分类与多类分类算法，比如线性SVM、逻辑回归、决策树、朴素贝叶斯。
- `org.apache.spark.mllib.clustering`：包含K均值聚类。
- `org.apache.spark.mllib.linalg`：包含数据表示，比如稠密向量、稀疏向量、矩阵。
- `org.apache.spark.mllib.optimization`：包含各种优化算法，在MLlib中用作底层原语，包含梯度下降算法、随机梯度下降法、分布式SGD更新方案、限制内存BFGS算法。
- `org.apache.spark.mllib.recommendation`：包含基于模型的协同过滤算法，使用交替最小二乘法的矩阵分解实现。
- `org.apache.spark.mllib.regression`：包含回归学习算法，比如线性最小二乘法、决策树、Lasso算法、岭回归。
- `org.apache.spark.mllib.stat`：包含样本（稀疏向量或稠密向量格式）统计函数，用来计算平均值、方差、最小值、最大值、向量个数与非零向量个数。
- `org.apache.spark.mllib.tree`：实现了分类与回归决策树学习算法。
- `org.apache.spark.mllib.util`：包含一系列对数据进行装载、预处理、生成与验证的方法。

### 2.2.5　Deeplearning4j

Deeplearning4j也称DL4J，是使用Java语言编写的深度学习库。它也是一个分布式或单机式的深度学习框架，包含并支持多种神经网络结构，比如前馈神经网络、**RBM**、卷积神经网络、深度信念网络、自编码器等。DL4J可以解决一些常见的识别问题，比如识别面部、声音以及垃圾邮件或电子商务欺诈。

Deeplearning4j在Apache 2.0许可证下发行，可以从http://deeplearning4j.org下载。Deeplearning4j

库的组织结构如下。

- org.deeplearning4j.base：包含加载类。
- org.deeplearning4j.berkeley：包含数学实用工具方法。
- org.deeplearning4j.clustering：包含K均值聚类算法的实现。
- org.deeplearning4j.datasets：操作数据集，允许的操作有导入、创建、遍历等。
- org.deeplearning4j.distributions：包含针对于分布的实用工具方法。
- org.deeplearning4j.eval：包含评估类，比如混淆矩阵。
- org.deeplearning4j.exceptions：实现了异常处理程序。
- org.deeplearning4j.models：包含监督学习算法，比如深度信念网络、栈式自编码器、堆叠式去噪自编码器、RBM。
- org.deeplearning4j.nn：实现了基于神经网络的组件与算法，比如神经网络、多层网络、卷积多层网络等。
- org.deeplearning4j.optimize：包含神经网络优化算法，比如反向传播算法、多层优化、输出层优化等。
- org.deeplearning4j.plot：包含各种绘制数据的方法。
- org.deeplearning4j.rng：一个随机数据生成器。
- org.deeplearning4j.util：包含辅助工具与实用方法。



### 2.2.6 MALLET

**MALLET**（**Machine Learning for Language Toolkit，自然语言机器学习工具集**）是一个大型库，包含自然语言处理算法与实用工具。它可以应用于各种任务，比如文本分类、文本聚类、信息抽取与主题建模等。对于一些算法，它提供了命令行接口与Java API，涉及的算法有朴素贝叶斯、HMM、**隐狄利克雷**主题模型、逻辑回归与条件随机场。




图2-5 MALLET

MALLET在通用公共许可证1.0下发行，这意味着你可以自由使用它，甚至可以用在商业应用中。可以从http://mallet. cs.umass.edu下载。MALLET实例用名称、标签、数据与源表示，然而可以使用如下两种方法将数据导入成MALLET格式。

- 每文件一实例：每个文件（即文档）对应一个实例，MALLET允许使用目录名作为输入。
- 每行一实例：每行对应一个实例，格式为：instance_name 标签标记。其数据是一个特征向量，由用作标记的不同词语与其出现的次数组成。

MALLET库包含如下包。

- `cc.mallet.classify`：包含用于训练与对实例分类的算法，比如AdaBoost、bagging、C4.5、其他决策树模型、多元逻辑回归、朴素贝叶斯与Winnow2。
- `cc.mallet.cluster`：包含无监督聚类算法，比如贪婪聚合（greedy agglomerative）、爬山算法、K-Best与K均值聚类。
- `cc.mallet.extract`：实现了分词器、文档提取器、文档查看器、清洁器（cleaners）等。
- `cc.mallet.fst`：实现了序列模型，包括条件随机场、HMM、最大熵马尔可夫模型与对应算法与评估器。
- `cc.mallet.grmm`：实现了图模型与因子图，比如推理算法、学习与测试。例如Loopy Belief Propagation、吉布斯采样等。
- `cc.mallet.optimize`：包含寻找函数最大值的优化算法，比如梯度上升法、限制内存BFGS、随机元上升法（stochastic meta ascent）等。
- `cc.mallet.pipe`：包含流水线方法，将数据处理为MALLET实例。
- `cc.mallet.topics`：包含主题建模算法，比如潜在狄利克雷分配（LDA）、Four-Level Pachinko Allocation、分层PAM、DMRT等。
- `cc.mallet.types`：实现了基础数据类型，比如数据集、特征向量、实例、标签。
- `cc.mallet.util`：包含其他杂项实用函数，比如命令行处理、搜索、数学、测试等。

### 2.2.7 比较各个库

表2-1总结并列出了目前所有库。表格不是很全面，除列出的这些库之外，还有许多针对特定问题域的其他库。表中列出的只是Java机器学习领域中一些比较有名的库。

表2-1 Java机器学习库

| | 问题域 | 许可证 | 架 构 | 算 法 |
|---|---|---|---|---|
| Weka | 通用用途 | GNU GPL | 单机 | 决策树、朴素贝叶斯、神经网络、随机森林、AdaBoost、层次聚类等 |
| Java-ML | 通用用途 | GNU GPL | 单机 | K均值聚类、自组织映射、Markov chain clustering、Cobweb、随机森林、决策树、bagging、距离度量等 |
| Mahout | 分类、推荐与聚类 | Apache 2.0 许可证 | 分布式，单机 | 逻辑回归、朴素贝叶斯、随机森林、HMM、多层感知器、K均值聚类等 |
| Spark | 通用用途 | Apache 2.0 许可证 | 分布式 | SVM、逻辑回归、决策树、朴素贝叶斯、K均值聚类、线性最小二乘法、LASSO、岭回归等 |
| DL4J | 深度学习 | Apache 2.0许可证 | 分布式，单机 | RBM、深度信念网络、深度自动编码器、递归神经张量网络（recursive neural tensor networks）、卷积神经网络、堆叠式去噪自编码器 |
| MALLET | 文本挖掘 | 通用公共许可证1.0 | 单机 | 朴素贝叶斯、决策树、最大熵、隐马尔可夫模型、条件随机场 |

## 2.3 创建机器学习应用

机器学习应用（特别是那些与分类相关的）通常遵循相同的高级工作流程，如图2-6所示。工作流程由两个阶段组成：训练分类器、对新实例做分类。正如你在图2-6中看到的那样，这两个阶段有一些相同步骤。



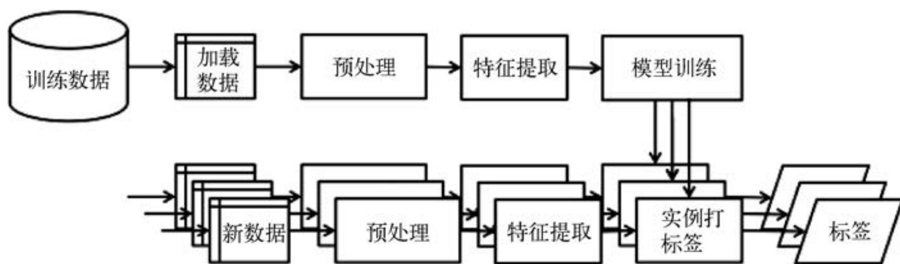



图2-6 机器学习应用工作流程

首先，使用一组**训练数据**，从中选取典型子集作为训练集，对缺失数据做预处理，提取特征。一个被选中的监督学习算法用于训练模型，第二个阶段将使用这个模型。第二个阶段中，先对新数据实例做**预处理**，再**提取特征**，然后应用训练好的模型得到实例标签。如果你能收集新的打过标签的数据，就可以周期性地再次运行学习阶段，重新训练模型，然后在分类阶段用新训练好的模型代替旧模型。

### 传统的机器学习架构

结构化数据（比如交易数据、客户数据、分析数据、市场数据）通常存储在本地关系数据库中。给出一门查询语言——比如SQL——我们就可以查询要处理的数据，就像前图工作流中显示的那样。通常，所有数据都能存储在内存中，并且可以使用机器学习库（比如Weka、Java-ML、MALLET）对它们做进一步处理。

架构设计中通常的做法是创建数据流水线，划分工作流中的不同步骤。比如，为了创建一条客户数据记录，我们可能必须从不同数据源采集数据，然后把记录保存到一个中间数据库，等待进一步处理。

为了理解大数据架构在高级层面上有何不同，我们先搞清楚什么样的数据才是大数据。

## 2.4 处理大数据

“大数据”这一术语出现之前，大数据存在已久，比如银行与证券交易所多年来每天处理的交易量有几十亿笔；航空公司拥有全球性的实时业务支撑设施，用来操作管理乘客预约信息等。

那么，大数据究竟是什么？Doug Laney（2001）提出用3个V定义大数据，即规模（volume）、快速（velocity）、多样（variety）。因此，回答你的数据“是不是大数据”的问题时，我们可以把这个问题分解成如下三个小问题。

- **规模（Volume）**：你的数据可以存储在内存中吗？
- **快速（Velocity）**：你可以用一台机器处理新输入的数据吗？
- **多样（Variety）**：你的数据源只有一个吗？

对于上面这三个问题，如果你的回答全是肯定的，那么你的数据很可能不是大数据。这样就没有什么可担心的了，因为这时你可以把应用程序架构设计得很简单。

如果你的回答全是否定的，那么你的数据就是大数据。然而，如果回答中既有肯定的也有否定的，那问题就有点复杂了。有些人可能认为“这个”V重要，而另一些人可能认为“那个”V更重要。从机器学习的观点看，处理内存中的数据与分布式存储中的数据时，两者的算法实现根本不同。因此，依据经验，通常的做法是：如果你不能把数据保存到内存，那么应该尝试使用大数据机器学习库。

这个问题的确切答案依赖于你尝试解决的问题。新项目启动之初，我建议你先从单机库（single-machine library）开始，如果不适合把所有数据放入内存，请尽可能先使用数据的子集为算法创建原型。一旦初步结果令人满意，你可以再考虑使用一些“重型”工具，比如Mahout或Spark。

## 大数据应用架构

大数据（比如文档、博客、社交网络、传感器数据等）通常存储在一个NoSQL数据库（比如MongoDB）或分布式文件系统（如HDFS）中。如果要处理结构化数据，可以使用Cassandra或HBase（建立在Hadoop上）等系统部署数据库功能。数据处理遵循MapReduce范式，它会把数据处理问题划分成更小的子问题，并把任务分配到各处理节点。机器学习模型最终要使用Mahout、Spark等机器学习库进行训练。



MongoDB是一个NoSQL数据库，采用与JSON类似的格式存储文档。更多内容请访问https://www.mongodb.org。

Hadoop是一个分布式基础框架，通过它可以充分利用计算机集群的能力，对大数据集进行分布式处理。Hadoop拥有自己的文件系统HDFS、作业调度框架YARD，并且针对并行数据处理实现了MapReduce方法。关于Hadoop的更多内容，请前往http://hadoop.apache.org/学习。

Cassandra是一个分布式数据库管理系统，拥有可容错、可扩展的分散存储数据的能力。更多信息请前往http://cassandra. apache.org/学习。

HBase也是一个数据库，它关注的重点是对分布式存储的随机读写访问。更多信息请前往https://hbase.apache.org/学习。

## 2.5　小结

你选择的机器学习库对你的应用程序架构有着重要影响，关键是考虑项目需求。你拥有什么样的数据？你试图解决什么样的问题？你的数据够大吗？你需要分布式存储吗？你打算使用哪种算法？一旦你搞清了解决问题时都需要些什么，接着就可以选择一个最适合你需求的库。

下一章将学习如何使用这些现成的库完成一些基本的机器学习任务，比如分类、回归、聚类。

第 3 章

# 基本算法——分类、回归和聚类

第2章学习了几个主要的Java机器学习库及其功能，本章我们将使用这些库，完成一些基本的机器学习任务。首先详细了解基本的机器学习任务，比如分类、回归、聚类。每一个主题都会介绍有关分类、回归、聚类的基本算法。本章给出的示例数据集小而简单，很容易理解。

本章涵盖主题如下：

- 加载数据
- 过滤属性
- 创建分类、回归、聚类模型
- 评估模型

## 3.1 开始之前

首先，从http://www.cs.waikato.ac.nz/ml/weka/downloading.html下载Weka 3.6最新版。

在页面上，你可以看到有多个下载项可用。我们想在源代码中将Weka用作库，所以跳过那些自解压可执行文件，在Other platforms (Linux, etc.)中点击here，下载包含Weka的ZIP文档，如图3-1所示。然后解压缩下载的文档，在解压后得到的文件夹中，找到weka.jar文件。

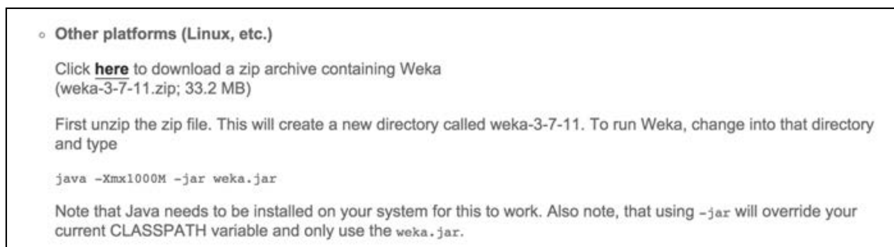



图3-1 Other platforms (Linux, etc.)页面

使用Eclipse IDE展现示例，如下：

(1) 启动一个新的Java项目；
(2) 右键点击项目属性，选择**Java Build Path**；点击**Libraries**选项卡，选择**Add External JARs**；
(3) 导航到解压后的文件夹，选择`weka.jar`文件。

上面就是需要准备的全部内容，下面实现基本的机器学习技术。



## 3.2　分类

我们将从最常用的机器学习技术——分类开始学起。正如第1章所讲，主要思想是在输入变量与输出结果之间自动创建一个映射。接下来，我们将学习如何加载数据、选择特征、实现一个基本的Weka分类器，以及对分类器的性能进行评估。

### 3.2.1　数据

要完成这项任务，先要看看`ZOO`数据库`[ref]`。这个数据库包含101个关于动物的数据项，每种动物使用18个属性进行描述，如表3-1所示。

表3-1　ZOO数据库

| | | |
|---|---|---|
| animal | aquatic | fins |
| hair | predator | legs |
| feathers | toothed | tail |
| eggs | backbone | domestic |
| milk | breathes | cat size |
| cat size | venomous | type |

从数据集中选择一个数据项——lion作为例子，其属性如下。

- `animal: lion`
- `hair: true`
- `feathers: false`
- `eggs: false`
- `milk: true`
- `airbone: false`
- `aquatic: false`
- `predator: true`
- `toothed: true`
- `backbone: true`

- breaths: true
- venomous: false
- fins: false
- legs: 4
- tail: true
- domestic: false
- domestic: false
- type: mammal

我们的任务是创建一个模型，通过输入的所有其他属性预测结果变量——animal。

### 3.2.2　加载数据

开始分析之前，先将数据加载为Weka的ARRF格式，并打印加载的实例总数。所有数据样本都包含在一个Instances对象中，带有元信息的完整数据集由该Instances对象进行处理。

加载输入数据之前，需要先创建DataSource对象。它可以接受各种文件格式，并将其转换成Instances。

```
DataSource source = new DataSource(args[0]);
Instances data = source.getDataSet();
System.out.println(data.numInstances() + " instances loaded.");
// System.out.println(data.toString());
```

加载的instances数量输出如下：

```
101 instances loaded.
```

此外，调用data.toString()方法，打印整个数据集。

我们的任务是学习创建模型，以便预测新样本的animal属性。对于这些新样本，我们只知道它们的一些其他属性，但不知道它们的animal属性。因此，从训练集中移除animal属性。为此，只要使用Remove过滤器将animal属性过滤即可。

首先，设置一个参数的字符串表，指定必须移走第一个属性。其余属性用作数据集，用来训练分类器。

```
Remove remove = new Remove();
String[] opts = new String[]{ "-R", "1"};
```

最后，调用Filter.useFilter(Instances, Filter)静态方法，将过滤器应用于所选数据集。

```
remove.setOptions(opts);
```

```
remove.setInputFormat(data);
data = Filter.useFilter(data, remove);
```

### 3.2.3 特征选择

第1章已经介绍过，预处理过程中，最重要的一个步骤是特征选择，也叫属性选择。它的目标是选择相关属性的一个子集，用在学习模型中。为什么特征选择如此重要呢？因为一个更小的属性集可以简化模型，并且让它们更容易被用户解释，这样通常会减少所需的训练时间，也会降低过拟合的风险。



做属性选择时，可以考虑类值，也可以不考虑。第一种情况下，可以使用属性选择算法评估特征的不同子集，并且计算出一个分数，表明所选属性的品质。我们可以使用不同的搜索算法（比如穷举搜索、最佳优先搜索）与不同的品质分数（比如信息增益、基尼指数等）。

Weka提供了一个`AttributeSelection`对象进行属性选择，它需要两个额外的参数：评价器（evaluator），用于计算属性的有用程度；排行器（ranker），用于根据评价者给出的分数对属性进行分类排序。

这个示例中，我们将把信息增益用作评价器，通过它们的信息增益分数对特征进行分类排序。

```
InfoGainAttributeEval eval = new InfoGainAttributeEval();
Ranker search = new Ranker();
```

接下来，对`AttributeSelection`对象进行初始化，设置评价器、排行器与数据。

```
AttributeSelection attSelect = new AttributeSelection();
attSelect.setEvaluator(eval);
attSelect.setSearch(search);
attSelect.SelectAttributes(data);
```

最后，将属性索引数组转换成字符串并打印如下：

```
int[] indices = attSelect.selectedAttributes();
System.out.println(Utils.arrayToString(indices));
```

输出结果如下：

```
12,3,7,2,0,1,8,9,13,4,11,5,15,10,6,14,16
```

最有价值的属性是`12`（fins）、`3`（eggs）、`7`（aquatic）、`2`（hair）等。基于这个结果，按次序剔除无用特征，让学习算法生成更准确、更快的学习模型。

那么，“要保留多少个属性”最后是由什么决定的呢？对于确切的数目，没有什么现成的经验可以借鉴，究竟保留多少属性取决于具体的数据与问题。属性选择的目的是选择那些可以让你的模型变得更好的属性，所以应该把重点放在考察“属性是否有助于进一步改进模型”上。

### 3.2.4　学习算法

加载数据并选好最佳特征后，接下来学习一些分类模型。先从最基本的决策树开始。

Weka中，决策树在J48类中实现，它重新实现了著名的Quinlan's C4.5决策树学习器（Quinlan，1993）。

首先，初始化一个新的J48决策树学习器。可以使用一个字符串表传递额外的参数，比如剪枝（tree pruning），它用来控制模型的复杂度（请参考第1章）。例子中，我们将创建一棵未剪枝树（un-pruned tree），为此传递一个U参数。

```
J48 tree = new J48();
String[] options = new String[1];
options[0] = "-U";

tree.setOptions(options);
```

接下来，调用buildClassifier(Instances)方法，对学习过程进行初始化。

```
tree.buildClassifier(data);
```

创建好的模型存储在tree对象中。可以调用toString()方法输出整个J48未剪枝树。

```
System.out.println(tree);
```

输出结果如下：

```
J48 unpruned tree
------------------

feathers = false
|   milk = false
|   |   backbone = false
|   |   |   airborne = false
|   |   |   |   predator = false
|   |   |   |   |   legs <= 2: invertebrate (2.0)
|   |   |   |   |   legs > 2: insect (2.0)
|   |   |   |   predator = true: invertebrate (8.0)
|   |   |   airborne = true: insect (6.0)
|   |   backbone = true
|   |   |   fins = false
|   |   |   |   tail = false: amphibian (3.0)
|   |   |   |   tail = true: reptile (6.0/1.0)
|   |   |   fins = true: fish (13.0)
|   milk = true: mammal (41.0)
feathers = true: bird (20.0)

Number of Leaves  : .9

Size of the tree : ..17
```

从输出结果可以看到，未剪枝树总共有17个节点，其中9个是叶子（Leaves）。

另一种呈现树的方法是利用内建的TreeVisualizer树浏览器，代码如下：

```
TreeVisualizer tv = new TreeVisualizer(null, tree.graph(), new PlaceNode2());
JFrame frame = new javax.swing.JFrame("Tree Visualizer");
frame.setSize(800, 500);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.getContentPane().add(tv);
frame.setVisible(true);
tv.fitToScreen();
```



代码执行结果如图3-2所示。

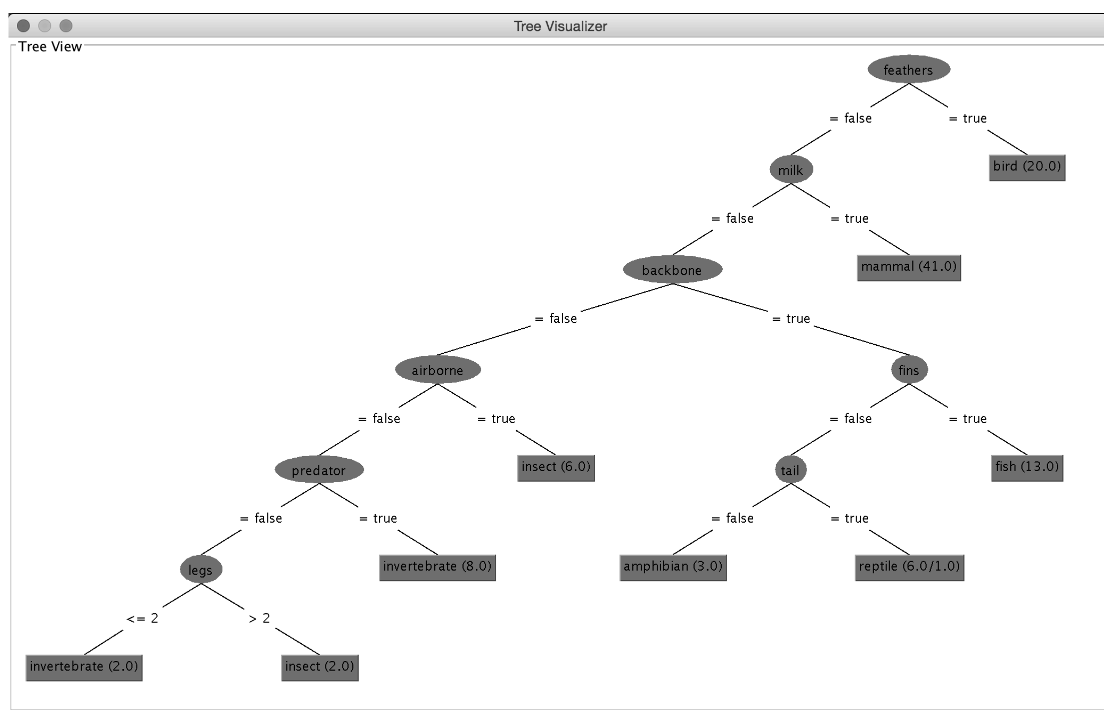



图3-2　代码执行结果

决策过程从顶部节点（也叫根节点）开始。节点标签指定要检查的属性值。示例中，先检查feathers属性的值。如果feathers属性值存在，进入右手分支，到达标记为bird的叶子，表示有20个样本支持这个输出结果。如果feathers属性值不存在，进入左手分支，到达下一个属性——milk，继续检查milk属性的值，然后进入与属性值相匹配的分支。不断重复这个过程，直到到达叶节点。

可以根据如下步骤创建其他分类器：初始化一个分类器，传递控制模型复杂度的参数，调用

buildClassifier(Instances)方法。

接下来，我们将学习如何使用一个经过训练的模型，为一个标签未知的新样本指派一个类标签。

### 3.2.5　对新数据分类

假如我们有一个动物的属性，但是不知道它的标签，此时可以使用已经学习的分类模型进行预测。

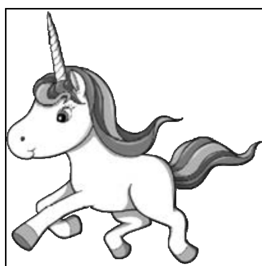


图3-3　标签未知的动物属性

首先，构造一个特征向量描述新样本，代码如下：

```
double[] vals = new double[data.numAttributes()];
vals[0] = 1.0;  //hair {false, true}
vals[1] = 0.0;  //feathers {false, true}
vals[2] = 0.0;  //eggs {false, true}
vals[3] = 1.0;  //milk {false, true}
vals[4] = 0.0;  //airborne {false, true}
vals[5] = 0.0;  //aquatic {false, true}
vals[6] = 0.0;  //predator {false, true}
vals[7] = 1.0;  //toothed {false, true}
vals[8] = 1.0;  //backbone {false, true}
vals[9] = 1.0;  //breathes {false, true}
vals[10] = 1.0;  //venomous {false, true}
vals[11] = 0.0;  //fins {false, true}
vals[12] = 4.0;  //legs INTEGER [0,9]
vals[13] = 1.0;  //tail {false, true}
vals[14] = 1.0;  //domestic {false, true}
vals[15] = 0.0;  //catsize {false, true}
Instance myUnicorn = new Instance(1.0, vals);
```

然后，调用模型的classify(Instance)方法获取类值，并返回标签索引，代码如下：

```
double result = tree.classifyInstance(myUnicorn);
System.out.println(data.classAttribute().value((int) result));
```

最后输出的结果是mammal类标签。

### 3.2.6 评估与预测误差度量

虽然创建了模型，但还不知道它是否值得我们信任。为了评估其性能，需要用到第1章讲到的交叉验证技术。

Weka提供`Evaluation`类，帮助我们实现交叉验证。使用时，需要提供模型、数据、折数（number of folds）以及一个初始的随机种子，代码如下：

```
Classifier cl = new J48();
Evaluation eval_roc = new Evaluation(data);
eval_roc.crossValidateModel(cl, data, 10, new Random(1), new
  Object[] {});
System.out.println(eval_roc.toSummaryString());
```



评估结果存储在`Evaluation`对象中。

调用`toString()`方法，显示最常用的指标评估结果：

```
Correctly Classified Instances          93               92.0792 %
Incorrectly Classified Instances         8                7.9208 %
Kappa statistic                          0.8955
Mean absolute error                      0.0225
Root mean squared error                  0.14
Relative absolute error                 10.2478 %
Root relative squared error             42.4398 %
Coverage of cases (0.95 level)          96.0396 %
Mean rel. region size (0.95 level)      15.4173 %
Total Number of Instances              101
```

在分类中，我们感兴趣的是正确分类的样本数与错误分类的样本数。

### 3.2.7 混淆矩阵

通过检查混淆矩阵，可以进一步查看特定的错误分类出现在什么地方。混淆矩阵指出了一个特定类值是如何进行预测的。

```
double[][] confusionMatrix = eval_roc.confusionMatrix();
System.out.println(eval_roc.toMatrixString());
```

上述代码产生的混淆矩阵如下：

```
=== Confusion Matrix ===

  a  b  c  d  e  f  g   <-- classified as
 41  0  0  0  0  0  0 |  a = mammal
  0 20  0  0  0  0  0 |  b = bird
  0  0  3  1  0  1  0 |  c = reptile
  0  0  0 13  0  0  0 |  d = fish
  0  0  1  0  3  0  0 |  e = amphibian
```

```
0  0  0  0  0  5  3 |  f = insect
0  0  0  0  0  2  8 |  g = invertebrate
```

第一行中，第一列的名称对应于分类模型指派的标签。然后，每一个附加行对应于一个实际为真的类值。比如，第二行对应于那些实际带有mammal类标签的实例。在列中，读取所有被正确分类为mammals的哺乳动物。第四行“爬行动物”中，可以看到有3个样本被正确分类为reptile，一个被分类为fish，一个被分类为insect。由此可见，混淆矩阵可以让我们进一步了解分类模型所犯错误的具体类型。

### 3.2.8　选择分类算法

机器学习中，朴素贝叶斯是一种最简单、最有效果且最有效率的归纳算法。特征独立（现实世界中很少有这种情况）时，从理论上来说，这是最好的。即使带有从属特征，它的性能也是非常好的（Zhang，2004）。主要缺点是它不能学习特征之间如何进行相互作用，比如，尽管你喜欢往茶里放些柠檬或牛奶，可你讨厌同时放入二者。

决策树的主要优点在于，模型是一棵树，它在样本的学习过程中很容易做解释与说明。决策树既可以处理名义特征，也可以处理数值特征，并且不用担心数据是否是线性可分的。

其他一些分类算法的例子如下。

- weka.classifiers.rules.ZeroR：预测多数类，并被看作基准线。如果你的分类器的性能比平均值预测器还差，就不值得考虑它。
- weka.classifiers.trees.RandomTree：构建一棵树，考虑每个节点上的$K$个随机选中的属性。
- weka.classifiers.trees.RandomForest：构建一组随机树（森林），并且使用多数投票算法对新实例进行分类。
- weka.classifiers.lazy.IBk：k最近邻分类器，它可以基于交叉验证选择一个合适的邻居值。
- weka.classifiers.functions.MultilayerPerceptron：基于神经网络的分类器，使用反向传播算法对实例进行分类。神经网络可以手工建立，也可以使用算法建立，或者两者兼用。
- weka.classifiers.bayes.NaiveBayes：朴素贝叶斯分类器，使用了估计器类（estimator classes），基于对训练数据的分析选择数值估计器的精确值。
- weka.classifiers.meta.AdaBoostM1：使用**AdaBoost M1**方法对名称类分类器进行增强，只处理名义类问题。在提升性能方面表现很突出，但有时会发生过度拟合问题。
- weka.classifiers.meta.Bagging：将多个基分类器组合在一起，以获得更为强大的分类器。可用于进行分类与回归，具体取决于基学习器（base learner）。

## 3.3　回归

本节将通过分析能源效率数据集（Tsanas和Xifara，2012）学习基本的回归算法。我们将基于建筑的结构特点（比如表面、墙体与屋顶面积、高度、紧凑度）研究它们的加热与冷却负载要求。研究者使用一个模拟器设计了12种不同的房屋配置，这些房屋配置通过改变18种建筑特征得出，他们总共模拟了768种建筑。

我们的首要目标是系统分析每种建筑特征对目标变量——加热或冷却负载——产生的影响。第二个目标是比较经典线性回归模型相对于其他方法（比如SVM回归、随机森林、神经网络）的性能。这个任务中，我们将使用Weka库。



### 3.3.1　加载数据

从https://archive.ics.uci.edu/ml/datasets/Energy+efficiency下载能源效率数据集。得到的数据集采用的是Excel的XLSX格式，Weka不能读取这样的格式。在Excel中打开数据集，单击“**文件-另存为**”；在弹出的“**另存为**”对话框中，将“**保存类型**”修改为CSV，将数据集转换为**CSV**（**Comma Separated Value**，**逗号分隔值**）格式文件，如图3-4所示。然后单击“**保存**”，在弹出对话框中，确保只存储当前工作表（其他两个工作表为空），单击“**确认**”；在弹出的“**如果另存为CSV**（**逗号分隔**），**您工作簿中的部分功能可能会丢失**”，单击“**是**”保存。现在，数据集已经转换为适合Weka加载的数据格式。

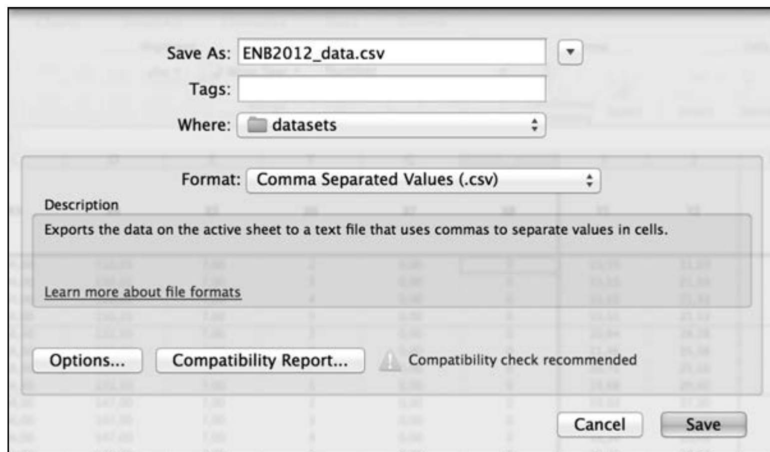



图3-4　将数据集转换为CSV格式文件

在文本编辑器中打开文件，检查文件转换是否正确。转换过程中可能出现一些小问题，它们会对后续数据处理带来潜在问题。比如，在我转换得到的文件中，每一行数据都以两个分号结尾，如下：

```
X1;X2;X3;X4;X5;X6;X7;X8;Y1;Y2;;
0,98;514,50;294,00;110,25;7,00;2;0,00;0;15,55;21,33;;
0,98;514,50;294,00;110,25;7,00;3;0,00;0;15,55;21,33;;
```

为了删除双分号，可以使用文本编辑器中的**查找与替换**功能，查找“;;”，并将其替换为“;”。

我遇到的另一个问题是，文档结尾有很多空行，如下所示，直接删除。

```
0,62;808,50;367,50;220,50;3,50;5;0,40;5;16,64;16,03;;
;;;;;;;;;;;
;;;;;;;;;;;
```

至此，我们已经准备好数据，接下来加载即可。新打开一个文件，使用Weka中的转换器编写一个简单的数据导入函数，读取CSV格式的文件。

```
import weka.core.Instances;
import weka.core.converters.CSVLoader;
import java.io.File;
import java.io.IOException;

public class EnergyLoad {

  public static void main(String[] args) throws IOException {

    // 加载CSV文件
    CSVLoader loader = new CSVLoader();
    loader.setSource(new File(args[0]));
    Instances data = loader.getDataSet();

    System.out.println(data);
  }
}
```

数据已经加载，继续下一步处理。

### 3.3.2　分析属性

进行属性分析之前，先了解要处理什么。总共有8个属性描述建筑特征，有两个目标变量：heating与cooling。

表3-2　建筑特征属性

| 属　　性 | 属性名称 |
|---|---|
| X1 | 相对密实性 |
| X2 | 表面积 |
| X3 | 墙体面积 |
| X4 | 屋顶面积 |

（续）

| 属　　性 | 属性名称 |
|---|---|
| X5 | 总体高度 |
| X6 | 方向 |
| X7 | 玻璃窗面积 |
| X8 | 玻璃窗区域分布 |
| Y1 | 加热负载 |
| Y2 | 冷却负载 |



### 3.3.3 创建与评估回归模型

首先，在特征位置设置分类属性，为加热负载建立学习模型。

```
data.setClassIndex(data.numAttributes() - 2);
```

第二个目标变量（冷却负载）现在可以移除：

```
//移除最后属性Y2
Remove remove = new Remove();
remove.setOptions(new String[]{"-R", data.numAttributes()+""});
remove.setInputFormat(data);
data = Filter.useFilter(data, remove);
```

#### 1. 线性回归

首先，使用LinearRegression类创建一个基本的线性回归模型。正如在分类示例中所做的那样，先初始化一个新模型实例，传递参数与数据，并调用buildClassifier(Instances)方法，代码如下：

```
import weka.classifiers.functions.LinearRegression;
...
data.setClassIndex(data.numAttributes() - 2);
LinearRegression model = new LinearRegression();
model.buildClassifier(data);
System.out.println(model);
```

最后学习的模型存储在model对象中，调用toString()方法进行输出，如下所示：

```
Y1 =

    -64.774  * X1 +
     -0.0428 * X2 +
      0.0163 * X3 +
     -0.089  * X4 +
      4.1699 * X5 +
     19.9327 * X7 +
      0.2038 * X8 +
     83.9329
```

线性回归模型构建了一个函数，它把输入变量线性组合在一起，对加热负载进行评估。特征前面的数字解释特征对目标变量的影响：符号表示正面影响或负面影响，而大小对应于影响程度。比如特征X1（相对紧凑度），它与加热负载是负相关的，而玻璃窗面积与加热负载是正相关的。这两个特征也对最后加热负载的评估有明显影响。

使用交叉验证技术可以对模型性能做类似评估。

做10折交叉验证（10-fold cross-validation）如下：

```
Evaluation eval = new Evaluation(data);
eval.crossValidateModel(
model, data, 10, new Random(1), new String[]{});
System.out.println(eval.toSummaryString());
```

这样即可输出常见的评估指标，包括相关系数、平均绝对误差、相对绝对误差等，如下所示：

```
Correlation coefficient                  0.956
Mean absolute error                      2.0923
Root mean squared error                  2.9569
Relative absolute error                 22.8555 %
Root relative squared error             29.282  %
Total Number of Instances              768
```

### 2. 回归树

另一个方法是构建一组回归模型，每一个模型对应于数据中与其自身相关的部分。图3-5展示了回归模型与回归树之间的主要不同。回归模型指的是一个与所有数据达到最好拟合的独立模型；而回归树是一组回归模型，每个模型只对一部分数据进行建模，如图3-5右图所示。相比于回归模型，回归树对数据的拟合更好，但函数在两块建模区域之间呈现线性分段现象。

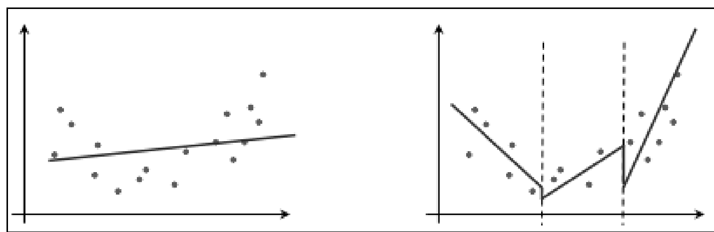


图3-5　回归模型与回归树的不同

Weka中的M5类用于实现回归树。创建模型时，遵从步骤与前面一样：初始化模型、传递参数与数据、调用buildClassifier(Instances)方法。

```
import weka.classifiers.trees.M5P;
...
M5P md5 = new M5P();
md5.setOptions(new String[]{""});
```

```
md5.buildClassifier(data);
System.out.println(md5);
```

该归纳模型是一棵叶节点带有方程式的树，如下：

```
M5经过剪枝的模型树:
(使用平滑线性模型)

X1 <= 0.75 :
|   X7 <= 0.175 :
|   |   X1 <= 0.65 : LM1 (48/12.841%)
|   |   X1 >  0.65 : LM2 (96/3.201%)
|   X7 >  0.175 :
|   |   X1 <= 0.65 : LM3 (80/3.652%)
|   |   X1 >  0.65 : LM4 (160/3.502%)
X1 >  0.75 :
|   X1 <= 0.805 : LM5 (128/13.302%)
|   X1 >  0.805 :
|   |   X7 <= 0.175 :
|   |   |   X8 <= 1.5 : LM6 (32/20.992%)
|   |   |   X8 >  1.5 :
|   |   |   |   X1 <= 0.94 : LM7 (48/5.693%)
|   |   |   |   X1 >  0.94 : LM8 (16/1.119%)
|   |   X7 >  0.175 :
|   |   |   X1 <= 0.84 :
|   |   |   |   X7 <= 0.325 : LM9 (20/5.451%)
|   |   |   |   X7 >  0.325 : LM10 (20/5.632%)
|   |   |   X1 >  0.84 :
|   |   |   |   X7 <= 0.325 : LM11 (60/4.548%)
|   |   |   |   X7 >  0.325 :
|   |   |   |   |   X3 <= 306.25 : LM12 (40/4.504%)
|   |   |   |   |   X3 >  306.25 : LM13 (20/6.934%)

LM num: 1
Y1 =
  72.2602 * X1
  + 0.0053 * X3
  + 11.1924 * X7
  + 0.429 * X8
  - 36.2224

...

LM num: 13
Y1 =
  5.8829 * X1
  + 0.0761 * X3
  + 9.5464 * X7
  - 0.0805 * X8
  + 2.1492

Number of Rules : 13
```

这棵树总共有13个叶子，每个叶子对应于一个线性方程。对预测输出做可视化处理，如图3-6所示。

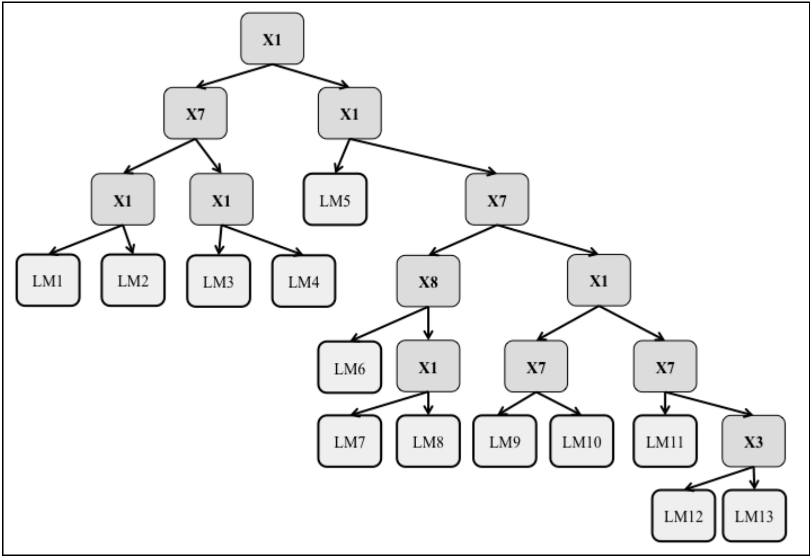



图3-6　对预测的可视化处理

这棵树的读取方法与分类树类似。最重要的特征在树顶。终端节点（叶子）包含一个线性回归模型，用于解释到达这部分的数据。

评估结果输出如下：

```
Correlation coefficient                  0.9943
Mean absolute error                      0.7446
Root mean squared error                  1.0804
Relative absolute error                  8.1342 %
Root relative squared error             10.6995 %
Total Number of Instances              768
```

## 3.3.4　避免常见回归问题的小技巧

首先，使用以前的研究与领域知识弄清哪些特征包含在回归中。查阅文献资料、报告与以前的研究，了解你的问题涉及哪些特征，以及可以使用哪些变量建模。假设你有大量关于随机数据的特征，但很可能只有几个特征与目标变量相关（尽管数据是随机的）。

要尽量让模型保持简单，防止过度拟合。奥卡姆剃刀原理告诉我们：应该在最少的假设下选择一个可以解释数据的最佳模型。事实上，模型可以简单到只有2~4个预测器特征。

## 3.4 聚类

相比于有监督的分类器，聚类的目标是从一组未打标签的数据中识别相似对象组。它可以用于识别同类群体的代表性样本，找到有用与合适的分组；或者找到不寻常的样本，比如异常值。

下面通过分析银行数据集演示如何实现聚类。数据集包含600个实例，每个实例用11个属性进行描述，这些属性包括年龄、性别、地区、收入、婚姻状况、是否有子女、汽车拥有情况、存款活动、当前活动、房地产抵押、PEP。分析中，我们将尝试使用**EM**（**Expectation Maximization**，**期望最大化**）聚类算法识别常见的客户组。

EM工作过程如下：给定一组簇（clusters），EM首先为每个实例指派一个属于某个特定簇的概率分布。比如，起初有3个簇A、B、C，一个实例分别属于簇A、B、C的概率分布依次为0.7、0.10、0.20。第二步中，EM重新评估每个类的概率分布的参数向量。算法不断对这两步做迭代，直到参数收敛或者达到迭代的最大值。

对于EM中使用的簇数，可以手动设置，也可以通过交叉验证进行自动设置。另外一个确定数据集中簇数的方法是肘部法则（elbow-method），这个方法会查看特定簇数所解释的偏差百分比。使用该方法会不断增加簇数，直到新加的簇不会带来很多信息，即只能解释很少的额外差异。

### 3.4.1 聚类算法

创建聚类模型的过程与创建分类模型的过程很类似，即加载数据与创建模型。Weka中，使用`weka.clusterers`包实现聚类算法，如下：

```
import java.io.BufferedReader;
import java.io.FileReader;

import weka.core.Instances;
import weka.clusterers.EM;

public class Clustering {

  public static void main(String args[]) throws Exception{

    // 加载数据
    Instances data = new Instances(new BufferedReader
      (new FileReader(args[0])));

    // 聚类器的新实例
    EM model = new EM();
    // 创建聚类器
    model.buildClusterer(data);
```

```
        System.out.println(model);

    }
}
```

该模型识别出如下6个簇：

```
EM
==

Number of clusters selected by cross validation: 6

                   Cluster
Attribute                0         1         2         3         4         5
                     (0.1)    (0.13)    (0.26)    (0.25)    (0.12)    (0.14)
==============================================================================
age
  0_34             10.0535   51.8472 122.2815   12.6207    3.1023    1.0948
  35_51            38.6282   24.4056  29.6252   89.4447   34.5208    3.3755
  52_max           13.4293     6.693   6.3459   50.8984    37.861   81.7724
  [total]          62.1111   82.9457 158.2526 152.9638   75.4841   86.2428
sex
  FEMALE           27.1812   32.2338  77.9304   83.5129   40.3199   44.8218
  MALE             33.9299   49.7119  79.3222   68.4509   34.1642    40.421
  [total]          61.1111   81.9457 157.2526 151.9638   74.4841   85.2428
region
  INNER_CITY       26.1651   46.7431   73.874   60.1973   33.3759   34.6445
  TOWN             24.6991   13.0716  48.4446   53.1731    21.617
17.9946
...
```

上表解读如下：第一行表明有6个簇，第一列指出属性及相应范围。比如，属性age划分为3个区段：0~34、35~51、52~max。左侧列表示有多少个实例归入每个簇的特定范围，比如0~34年龄组的客户大部分位于簇#2（122个实例）。

### 3.4.2　评估

可以使用对数似然度量（log likelihood measure）评估聚类算法的质量，即测量被识别的簇的一致程度。数据集划分为多个折（folds），针对每个折运行聚类。这么做的动机是，如果聚类算法为相似数据（该数据不用于拟合参数）给出高概率，那么它在捕获数据结构方面可能做得很好。Weka提供CluterEvaluation类进行评估，代码如下：

```
double logLikelihood = ClusterEvaluation.crossValidateModel(model, data, 10, new
Random(1));
System.out.println(logLikelihood);
```

输出结果如下：

```
-8.773410259774291
```

## 3.5 小结

本章学习了如何使用Weka实现基本的机器学习任务：分类、回归、聚类，并且对属性选择过程、训练模型与评估模型性能做了简短的讨论。

下一章将主要讲解如何使用这些技术解决实际问题，比如客户维系（customer retention）问题。

第 4 章

# 利用集成方法预测客户关系

不论哪类公司，无论是提供服务、卖产品，还是兜售经验，都需要深刻理解他们与客户之间的关系。因此，做好**客户关系管理**（**CRM**）就成为现代营销策略的关键。当今商业面临的最大挑战之一就是准确了解刺激顾客购买新产品的因素。

本章将以一个真实的营销数据库——Orange作为示例，它由法国电信公司提供。我们的任务是针对客户行为做如下评估：

- 更换供应商（客户流失）
- 购买新产品或服务（购物欲望）
- 购买升级或向客户推荐购买，从销售中赚取更多利润（追加销售）

我们将一起尝试解决**KDD**（**知识发现与数据挖掘**）Cup 2009挑战赛中的问题，并展示使用Weka处理数据的步骤。首先，解析与加载数据，并实现基本的基准模型（baseline models）。然后讲解高级建模技术，包括数据预处理、属性选择、模型选择与评估。



KDD Cup是世界最著名的数据挖掘竞赛，由ACM **SIGKDD**（**Special Interest Group on Knowledge Discovery and Data Mining**）每年组织一次。最终冠军会在**Conference on Knowledge Discovery and Data Mining**进行公布，通常在8月。

历年文档——包括所有相应数据集——都可以在http://www.kdd.org/kdd-cup上找到。

## 4.1　客户关系数据库

建立有关客户行为的知识时，最实用的方法是生成分数，用来解释目标变量，比如流失量、购买欲望或追加销售。分数由模型计算得到，计算时需要用到描述客户的输入变量，比如当前订阅、已购买设备、消耗时间等。这些分数之后会被信息系统使用，比如提供相关的个性化营销行为。

2009年，KDD大会组织了一场关于客户关系预测的机器学习挑战赛。

### 4.1.1 挑战

给定大量客户属性，对下面三个目标变量进行评估（KDD Cup，2009）。

- **流失概率**（**Churn probability**）：我们的示例中，流失概率指客户更换供应商的可能性。流失率（Churn rate）有时也叫损耗率（attrition rate）。它是决定一门生意所拥有的客户稳定水平的两个主要因素之一。从广义上说，流失率衡量某个特定时期内个人或物品移入或移出某个集合的数目。这个术语应用于许多领域，但在商业领域应用最广泛，与契约式客户群有关。比如，不管哪门生意，拥有一个基于订阅用户的服务模型是非常重要的，包括移动通信网络与付费电视运营商。这个术语也指对等网络中的参与者成交量（participant turnover）。
- **购买概率**（**Appetency probability**），我们的示例中，它指用户购买一项服务或产品的倾向。
- **追加销售概率**（**Upselling probability**），指一个客户购买附属商品或升级购买的可能性。追加销售（Upselling）是一项销售技术，借此，销售人员尝试让客户购买更昂贵的商品、升级购买或购买其他附属商品，谋求更多销售利润。追加销售通常指向客户推销更有利可图的服务或产品，但也可以指简单地把那些客户之前未考虑购买的商品展现给他们。追加销售暗指销售一些额外的商品，或销售那些更赚钱的商品；或者相反，销售卖方更希望卖出的商品，而非销售原来那些商品。

这项挑战赛在Orange Labs开发的内部系统中展开。对于那些参加者而言，这是一个难得机会，他们可以借此证明自己有能力处理大型数据库，包括混合的噪声数据与不平衡的类分布。

### 4.1.2 数据集

这次挑战赛中，Orange公司提供了一个关于客户数据的大型数据集，包含大约100万个客户，使用了10个表格与几百个字段进行描述。第一步中，他们对数据重新抽样，选择一个较平衡的子集，包含100 000个客户。第二步中，他们使用一个自动特征构建工具生成20 000个特征描述客户，然后把特征数减少到15 000个。第三步中，将特征顺序随机打乱，丢弃属性名，使用随机生成的字符串替换名义变量，用一个随机因子乘以连续属性，对数据集做匿名化处理。最后，把所有实例随机划分为训练数据集与测试数据集。

KDD Cup提供了两组数据：大型数据集与小型数据集，依次对应于快速挑战与慢速挑战。在KDD Cup网站中描述如下：

> 训练集与测试集包含50 000个样本。数据被类似地划分为小型版本和大型版本，但在训练集与测试集中，样本的排列顺序不同。小型数据集与大型数据集都有数值与分类变量。就大数据集而言，前14 740个变量是数值型，后面260个是分类型。对于小型数据集而言，前190个变量是数值型，后面40个是分类型。

本章将使用小型数据集，它包含50 000个实例，每个实例用230个变量进行描述。数据集有50 000行数据，每行对应于一个客户，以及3个二元结果（binary outcomes），其中每一个对应于3个挑战项（追加销售、流失、购买欲）之一。

为了给大家一个更清晰的认识，表4-1描述了数据集的构成。

表4-1　数据集构成

| 230个数值与名义属性 | | | | | | | | | | 3个二元类 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Var85 | Var123 | Var125 | Var126 | Var132 | Var133 | Var134 | Var225 | Var229 | Var230 | 流失标签 | 购买欲标签 | 追加销售标签 |
| 12 | 6 | 720 | 8 | 0 | 1212385 | 69134 | | | | -1 | -1 | -1 |
| 2 | 72 | 0 | | 8 | 4136430 | 357038 | | | | 1 | -1 | -1 |
| 58 | 114 | 5967 | -28 | 0 | 3478905 | 248932 | kG3k | am7c | | -1 | -1 | -1 |
| 0 | 0 | 0 | -14 | 0 | 0 | 0 | | | | -1 | -1 | -1 |
| 0 | 0 | 15111 | 58 | 0 | 150650 | 66046 | kG3k | mj86 | | -1 | -1 | -1 |
| 10 | 0 | 1935 | | 8 | 641020 | 43684 | | am7c | | -1 | -1 | -1 |
| 16 | 24 | 13194 | -24 | 0 | 1664450 | 104978 | kG3k | am7c | | -1 | -1 | -1 |
| 2 | 12 | 0 | -8 | 8 | 3839825 | 1284128 | | | | -1 | -1 | -1 |
| 2 | 90 | 2754 | | 0 | 3830510 | 203586 | kG3k | am7c | | -1 | -1 | -1 |
| 24 | 66 | 6561 | | 32 | 2577245 | 210014 | kG3k | | | -1 | -1 | -1 |
| 6 | 12 | 5823 | 58 | 0 | 0 | 7134 | kG3k | mj86 | | -1 | -1 | -1 |
| 28 | 24 | 66825 | 52 | 8 | 134105 | 15166 | kG3k | | | -1 | -1 | -1 |
| 0 | 0 | 44154 | 10 | 0 | 0 | 0 | | mj86 | | -1 | -1 | -1 |
| 22 | 54 | 5202 | | 0 | 2772010 | 1095062 | xG3x | | | -1 | -1 | -1 |
| 0 | 102 | 31104 | 8 | 0 | 2170355 | 57596 | | | | -1 | -1 | 1 |
| 0 | 0 | 2574 | | 0 | 0 | 0 | ELof | oJmt | | -1 | -1 | -1 |
| 14 | 186 | 8019 | | 48 | 3571845 | 587392 | kG3k | am7c | | -1 | -1 | -1 |
| 0 | 30 | 5319 | | 8 | 500295 | 31436 | | am7c | | -1 | -1 | -1 |
| 2 | 0 | 13788 | 4 | 0 | 918350 | 0 | kG3k | | | -1 | -1 | -1 |
| 14 | 0 | 7110 | | 0 | 2055150 | 392138 | | | | 1 | -1 | -1 |
| 8 | 66 | 0 | -8 | 0 | 3258940 | 1121306 | | | | -1 | -1 | -1 |
| 0 | 18 | 0 | -10 | 0 | 0 | 0 | | | | -1 | -1 | -1 |
| 12 | 0 | 531 | 36 | 0 | 491345 | 56742 | ELof | mj86 | | -1 | -1 | -1 |
| 0 | 12 | 16803 | 12 | 0 | 201110 | 1693090 | | | | 1 | -1 | -1 |
| 14 | 0 | 25740 | | 0 | 2932660 | 313200 | xG3x | | | -1 | -1 | 1 |

上表中列出了前25个客户实例，每个实例使用230个属性进行描述，表格只选取了其中10个属性进行展示。数据集包含许多缺失值，有些属性甚至是空值或常数。表格中的最后3列对应于3个不同的类标签，用于表明客户是否真的更换了供应商（流失）、购买了一项服务（购买欲），或做了购买升级（追加销售）。请注意，3个不同文件中，标签与数据是分开提供的，所以有必要把实例顺序与类标签恰当地对应起来。

## 4.1.3　评估

针对三个任务（流失、购买欲、追加销售），根据ROC曲线下面区域面积的算术平均值对提

交的方案进行评估。ROC曲线使用一条曲线表示模型性能，这条曲线是通过为各种阈值（确定分类结果）描绘敏感性与特异性而得到的（请参考第1章中有关ROC曲线的内容）。AUC指ROC曲线下面区域的面积，面积越大，分类器越好。包含Weka在内的大多数工具箱都提供用于计算AUC分数的API。

## 4.2　最基本的朴素贝叶斯分类器基准

按照挑战规则，参与者必须胜过最基本的朴素贝叶斯分类器才有资格获奖。朴素贝叶斯分类器基于一个简单的假设：属性之间相互独立（参见第1章相关内容）。

KDD Cup组织者运行朴素贝叶斯分类器，不选择任何特征或对超参数做调整。对于大型数据集，朴素贝叶斯分类器在测试集上的综合分数如下：



- 客户流失问题：AUC = 0.6468
- 购买欲问题：AUC = 0.6453
- 追加销售问题：AUC=0.7211



请注意，基准结果只针对大型数据集。而且，KDD Cup网站上都提供了训练数据集与测试数据集，但并没有提供测试集的真实标签。因此，使用我们自己的模型处理数据时，无法得知模型在测试集上的工作效果。我们只能使用训练数据，并使用交叉验证评估模型。最终得到的结果可能没有直接可比性，但可以借此了解AUC分数的合理大小。

### 4.2.1　获取数据

在KDD Cup网页（http://kdd.org/kdd-cup/view/kdd-cup-2009/Data）中，你应该能够看到图4-1所示的数据页面。首先，从**Small version (230 var.)**部分下载`orange_small_train.data.zip`。接着，下载与该训练数据相关的3组真实标签。在**Real binary targets (small)**部分，你会看到如下文件：

- `orange_small_train_appentency.labels`
- `orange_small_train_churn.labels`
- `orange_small_train_upselling.labels`

保存并解压缩图4-1框中的所有文件。

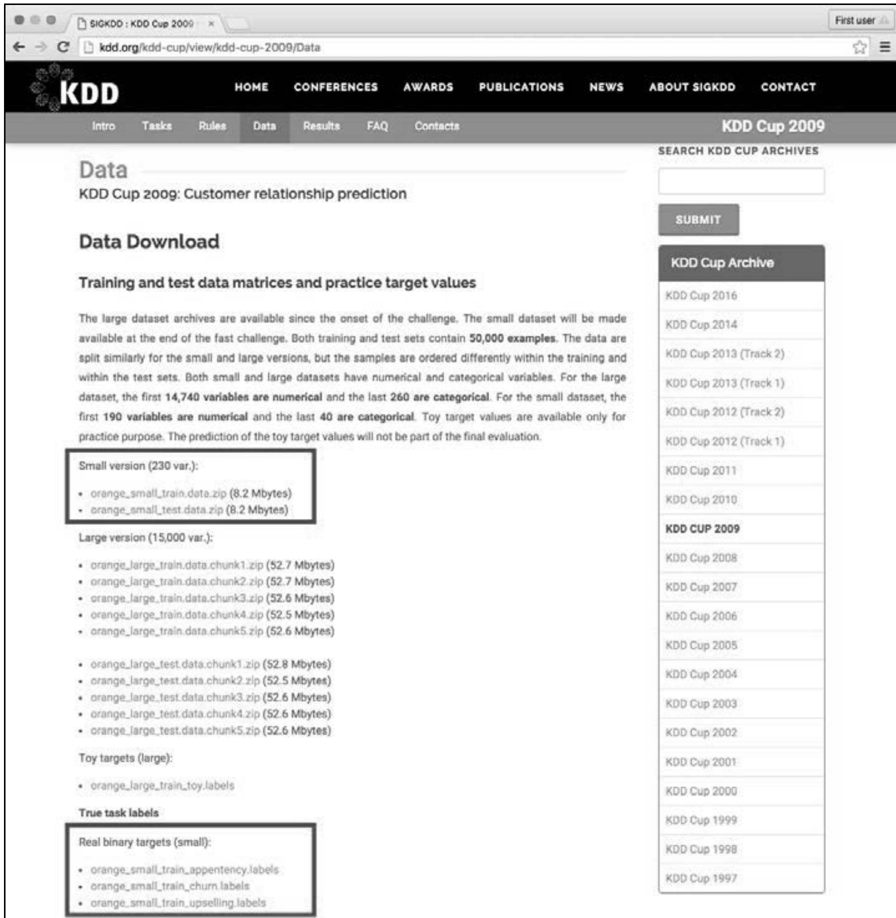



图4-1　KDD Cup数据页面

接下来的部分中，首先将数据加载到Weka，使用朴素贝叶斯方法建立基本模型，获得我们自己的基准AUC分数。然后学习更多关于高级建模技术与技巧的内容。

## 4.2.2　加载数据

直接将`.cvs`格式的数据加载到Weka。为此，编写一个加载数据的函数，它拥有两个参数，分别用于接收数据文件路径与真实标签文件路径。这个函数将加载并合并数据集，并且移走数据集中的空属性。

```
public static Instances loadData(String pathData, String
  pathLabeles) throws Exception {
```

首先，使用`CSVLoader`类加载数据。并且，指定`\t`为字段分隔符，将最后40个属性强制解

析为Nominal类型。

```
// 加载数据
CSVLoader loader = new CSVLoader();
loader.setFieldSeparator("\t");
loader.setNominalAttributes("191-last");
loader.setSource(new File(pathData));
Instances data = loader.getDataSet();
```



CSVLoader类还可以接收许多其他参数，用于指定列分隔符、字符串闭包，以及是否有标题行等。有关CSVLoader类的完整文档，可以在如下网址找到：http://weka.sourceforge.net/doc.dev/weka/core/converters/CSVLoader.html



接下来，一些属性不包含单个值，Weka自动将其识别为String属性。其实我们并不需要它们，所以可以使用RemoveType过滤器将其安全剔除。另外，我们指定了-T参数，它表示移走特定类型的属性与我们想移除的属性类型。

```
//将识别为String属性的空属性移走
RemoveType removeString = new RemoveType();
removeString.setOptions(new String[]{"-T", "string"});
removeString.setInputFormat(data);
Instances filteredData = Filter.useFilter(data, removeString);
```

或者也可以使用Instances类中的void deleteStringAttributes()方法进行同样的移除工作，比如data.deleteStringAttributes()。

接下来，加载类标签（class labels），并将其指派给数据。为此，要再次利用CVSLoader类，并且指定文件不带有任何标题行，即setNoHeaderRowPresent(true)。

```
// 加载标签
loader = new CSVLoader();
loader.setFieldSeparator("\t");
loader.setNoHeaderRowPresent(true);
loader.setNominalAttributes("first-last");
loader.setSource(new File(pathLabeles));
Instances labels = loader.getDataSet();
```

加载好两个文件后，可以调用Instances. mergeInstances (Instances, Instances)静态方法将其合并。该方法会返回一个新的数据集，里面包含来自第一个数据集与第二个数据集的所有属性。请注意，两个数据集中的实例个数必须一样。

```
// 将标签添加为类值
Instances labeledData = Instances.mergeInstances(filteredData, labeles);
```

设置最后一个属性，它是我们刚刚添加的标签属性，用作目标变量，然后返回最终数据集。

```
// 将标签属性设置为类别
labeledData.setClassIndex(labeledData.numAttributes() - 1);
```

```
System.out.println(labeledData.toSummaryString());
return labeledData;
}
```

最终函数输出结果如下，返回带标签的数据集：

```
Relation Name:  orange_small_train.data-weka.filters.unsupervised.attribute.
RemoveType-Tstring_orange_small_train_churn.labels.txt
Num Instances:  50000
Num Attributes: 215

Name          Type  Nom  Int Real     Missing      Unique  Dist
1 Var1        Num   0%   1%   0% 49298 / 99%     8 /  0%    18
2 Var2        Num   0%   2%   0% 48759 / 98%     1 /  0%     2
3 Var3        Num   0%   2%   0% 48760 / 98%   104 /  0%   146
4 Var4        Num   0%   3%   0% 48421 / 97%     1 /  0%     4
...
```

## 4.3　基准模型

这一部分将使用KDD Cup组织者采用的方法，实现我们自己的基准模型。但开始创建模型之前，先实现评估引擎，它会返回3个问题的AUC。

### 4.3.1　评估模型

下面详细了解评价函数。评价函数接收一个初始化的模型，对3个问题的模型进行交叉验证，并返回结果。该结果是AUC，如下：

```
public static double[] evaluate(Classifier model)
  throws Exception {

  double results[] = new double[4];

  String[] labelFiles = new String[]{
    "churn", "appetency", "upselling"};

  double overallScore = 0.0;
  for (int i = 0; i < labelFiles.length; i++) {
```

首先调用`Instance loadData(String, String)`函数，该函数在前面已经实现，用于加载训练数据，并合并训练数据与所选标签。

```
    //加载数据
    Instances train_data = loadData(
     path + "orange_small_train.data",
      path+"orange_small_train_"+labelFiles[i]+".labels.txt");
```

接下来，初始化weka.classifiers.Evaluation类，并传入我们的数据集（该数据集只提取数据属性而不考虑实际数据）。然后调用void crossValidateModel(Classifier, Instances, int, Random)方法，开始交叉验证，选择并生成5个折（five folds）。由于验证基于数据的随机子集进行，所以也需要我们传入一个随机种子。

```
// 对数据做交叉验证
Evaluation eval = new Evaluation(train_data);
eval.crossValidateModel(model, train_data, 5,
new Random(1));
```

评估完成后，调用double areaUnderROC(int)方法读取结果。由于度量依赖于我们感兴趣的目标值，所以该方法需要一个类值索引，通过在类属性中搜索值1的索引即可提取。

```
    // 保存结果
    results[i] = eval.areaUnderROC(
      train_data.classAttribute().indexOfValue("1"));
    overallScore += results[i];
  }
```



最后，求结果的平均值并返回。

```
  // 获得3个问题的平均结果
  results[3] = overallScore / 3;
  return results;
}
```

### 4.3.2 实现朴素贝叶斯基准线

拥有所有材料后，可以实现朴素贝叶斯方法，它是我们希望超越的目标。这个方法不会包含其他任何数据预处理、属性选择、模型选择。由于我们没有测试数据的实际标签，所以使用5折交叉验证（five-fold cross validation）在小数据集上评估模型。

首先，初始化朴素贝叶斯分类器，如下：

```
Classifier baselineNB = new NaiveBayes();
```

接着，将分类器传给评价函数，评估函数加载数据与应用交叉验证，并且为3个问题返回AUC分数以及总体结果：

```
double resNB[] = evaluate(baselineNB);
System.out.println("Naive Bayes\n" +
"\tchurn:     " + resNB[0] + "\n" +
"\tappetency: " + resNB[1] + "\n" +
"\tup-sell:   " + resNB[2] + "\n" +
"\toverall:   " + resNB[3] + "\n");
```

我们的示例中，模型返回的结果如下：

```
Naive Bayes
  churn:     0.5897891153549814
  appetency: 0.630778394752436
  up-sell:   0.6686116692438094
  overall:   0.6297263931170756
```

我们使用更先进的模型迎接挑战时，上面这些结果用作基准线。如果使用更高级、更耗时与更复杂的技术处理数据，就能得到更好的结果，否则只是在浪费资源。一般来说，解决机器学习问题时，创建一个简单的基准分类器用作定位点总是个不错的主意。

## 4.4　使用集成方法进行高级建模

前面实现了一个定位基准，接下来让我们把关注点放在“重型机械”上。我们将采用KDD Cup 2009获奖方案（由**IBM Research**团队开发，Niculescu-Mizil等，2009）使用的方法。

应对挑战过程中，IBM Research团队使用的是**集成选择**（**Ensemble Selection**）算法（Caruana与Niculescu-Mizil，2004）。它是一个集成方法，创建了一系列模型，并且按照特定方式将其输出组合在一起，形成最终分类结果。这个方法拥有几个优点，使得它非常适用于此次挑战。

- 这个方法非常健壮，并且拥有良好性能，这些优点已经得到证实。
- 它可以针对特定性能指标做优化，包括AUC。
- 它允许将不同分类器添加到库。
- 它是一个任何时候都适用的方法，这意味着，如果我们用完了时间，仍有解决方案可用。

本节大致根据他们在汇报中描述的步骤进行。请注意，我们并非要准确实现他们的方法，而只实现方案的大概，包括必需的深入步骤。

步骤概述如下：

(1) 首先，对数据做预处理，剔除没有价值的属性，比如所有缺失值与常数值。修复缺失值以帮助机器学习算法对其进行处理，将分类属性转换为数值；

(2) 接着，运行属性选择算法，选择属性的一个子集，用在任务预测中；

(3) 第三步，使用多种模型将集成选择算法实例化，最后评估其性能。

### 4.4.1　开始之前

这个任务中，我们需要用到另外一个Weka包——`ensembleLibrary`。Weka 3.7.2或更高版本支持大部分学术社区开发的外部包。http://weka.sourceforge.net/packageMetaData页面列出了可用的**WEKA**包，如图4-2所示。

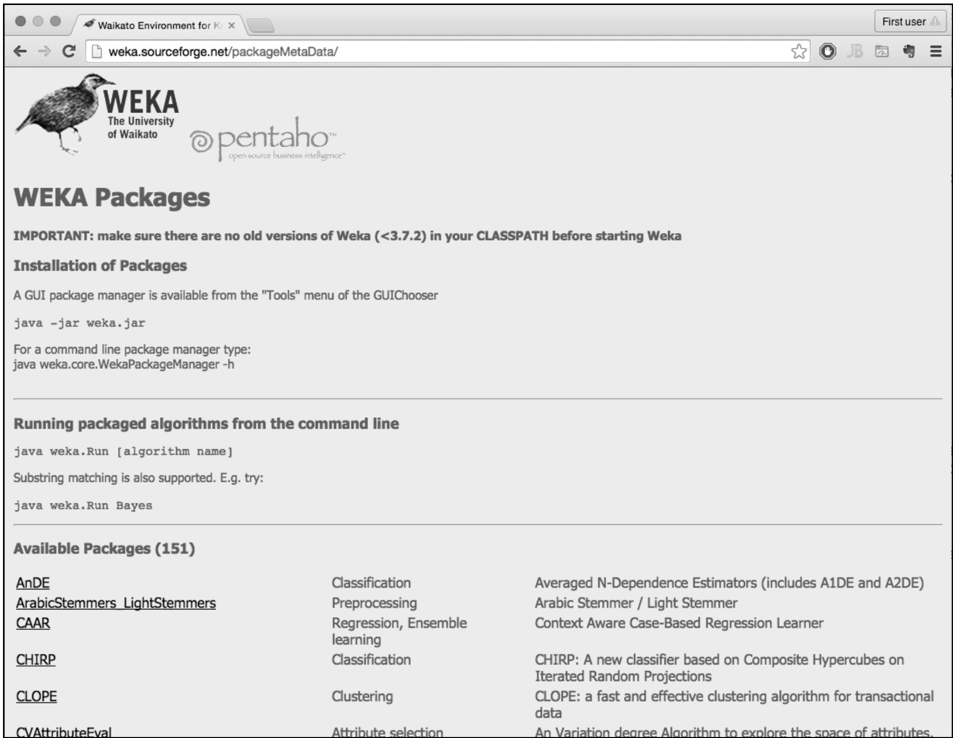



图4-2 可用的WEKA包

在http://prdownloads.sourceforge.net/weka/ensembleLibrary1.0.5.zip?download页面找到并下载最新版本的`ensembleLibrary`包。

对`ensembleLibrary`包进行解压缩后，找到`ensembleLibrary.jar`文件，并将其导入你的代码，如下所示：

```
import weka.classifiers.meta.EnsembleSelection;
```

## 4.4.2 数据预处理

首先，使用Weka内置的`weka.filters.unsupervised.attribute.RemoveUseless`过滤器移除无用属性。借助该过滤器，可以从数据中移除那些变化不大的属性，比如所有常量属性，并且移除那些变化太过剧烈（变化接近随机）的属性。可以通过`-M`参数指定最大方差（maximum variance），它只应用于名义属性（nominal attributes）。其默认值是99%，这意味着，对于某个属性，如果超过99%的实例拥有唯一的属性值，那么该属性就会被移除，如下所示：

```
RemoveUseless removeUseless = new RemoveUseless();
removeUseless.setOptions(new String[] { "-M", "99" });// 阈值
```

```
removeUseless.setInputFormat(data);
data = Filter.useFilter(data, removeUseless);
```

接下来，通过weka. filters.unsupervised.attribute.ReplaceMissingValues过滤器使用从训练数据得到的众数（名义属性）与平均数（数值属性），以此代替数据集中的所有缺失值。一般来说，对缺失值做替换处理时要多加小心，同时也要考虑属性的含义与上下文环境。

```
ReplaceMissingValues fixMissing = new ReplaceMissingValues();
fixMissing.setInputFormat(data);
data = Filter.useFilter(data, fixMissing);
```

最后，对数值型属性做离散化处理，也就是使用weka.filters.unsupervised.attribute.Discretize过滤器将数值型属性变换为区间（intervals）。使用-B选项，将数值型属性划分为4个区间，使用-R选项指定属性范围（只有数值型属性会被离散化）：

```
Discretize discretizeNumeric = new Discretize();
discretizeNumeric.setOptions(new String[] {
    "-B",  "4",  // 区间数
    "-R",  "first-last"}); //属性范围
fixMissing.setInputFormat(data);
data = Filter.useFilter(data, fixMissing);
```

### 4.4.3　属性选择

下面只选择包含有用信息的属性，也就是说，选择那些可能对预测更有帮助的属性。这个问题的标准做法是，检查每个属性包含的信息增益。我们将使用weka.attributeSelection.AttributeSelection过滤器进行属性选择，它需要另外两个方法：一个是评估器，指出如何计算属性的有用性；另一个是搜索算法，指出如何选择属性子集。

我们的示例中，先初始化weka.attributeSelection.InfoGainAttributeEval，它实现了对信息增益的计算。

```
InfoGainAttributeEval eval = new InfoGainAttributeEval();
Ranker search = new Ranker();
```

为了从高于某个阈值的属性中只选择顶级属性，初始化weka. attributeSelection.Ranker，对高于特定阈值且带有信息增益的属性进行排列。使用-T参数进行指定，同时保持低阈值，让属性至少带有一些信息。

```
search.setOptions(new String[] { "-T", "0.001" });
```



设置阈值的一般规则是，根据信息增益对属性进行排序，然后选择信息增益降到微不足道时的阈值。

接下来，初始化AttributeSelection类，设置评估器（evaluator）与排行器（ranker），并

且向数据集应用属性选择。

```
AttributeSelection attSelect = new AttributeSelection();
attSelect.setEvaluator(eval);
attSelect.setSearch(search);

// 应用属性选择
attSelect.SelectAttributes(data);
```

最后，调用`reduceDimensionality(Instances)`方法，移除上次运行未被选中的那些属性。

```
// 移除上次运行未被选中的属性
data = attSelect.reduceDimensionality(data);
```

最终，从230个属性中保留了214个属性。



### 4.4.4　模型选择

多年来，活跃在机器学习领域的实践者们开发了各种各样的学习算法，并对一些现有算法做了大量改进。独特的监督学习方法数量众多，很难全部记录。由于数据集的特征各不相同，所以不可能有适用于所有情况的通用方法。不同算法可以利用一个给定数据集的不同特征与关系。集成选择算法的特征就是尝试充分利用各种算法分析数据集（Jung，2005）。

> 直观地说，集成选择算法的目标是自动检测并组合这些独特算法的力量，使整体效果好于其中任何一个。这通过创建一个库来实现，其目的是尽可能多地利用这些不同的学习方法。这种过量生成大量模型的范式与传统的集成方法有很大不同。到目前为止，我们的结果非常鼓舞人心。

首先需要创建模型库。通过初始化`weka.classifiers. EnsembleLibrary`类实现，这个类会帮助我们定义模型。

```
EnsembleLibrary ensembleLib = new EnsembleLibrary();
```

接着，使用字符串向库添加模型及参数。比如使用不同参数向库添加两个决策树学习器，代码如下：

```
ensembleLib.addModel("weka.classifiers.trees.J48 -S -C 0.25 -B -M
  2");
ensembleLib.addModel("weka.classifiers.trees.J48 -S -C 0.25 -B -M
  2 -A");
```

如果熟悉Weka图形界面，你也可以用它找到这两个算法及其配置，然后复制算法配置：在算法名称上点击鼠标右键，依次选择**Edit configuration>Copy configuration string**，如图4-3所示。

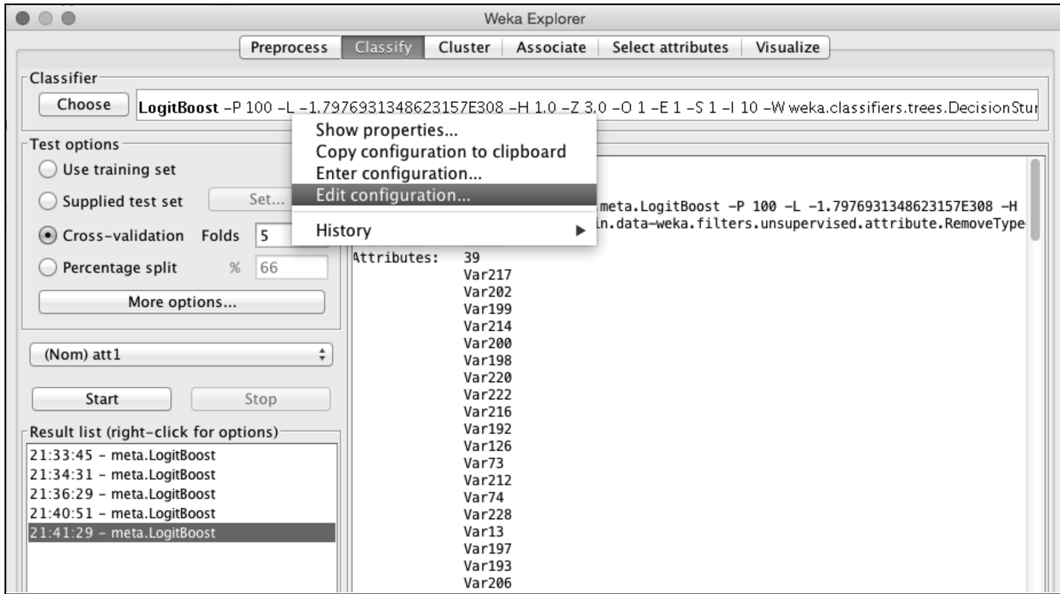



图4-3　Weka算法配置页面

为了完成示例，继续添加如下算法及其参数。

- 朴素贝叶斯算法，用作默认基准：

```
ensembleLib.addModel("weka.classifiers.bayes.NaiveBayes");
```

- 基于懒惰模型（lazy models）的k最近邻算法：

```
ensembleLib.addModel("weka.classifiers.lazy.IBk");
```

- 逻辑回归，用作简单逻辑，带有默认参数：

```
ensembleLib.addModel("weka.classifiers.functions.SimpleLogi stic");
```

- 带有默认参数的支持向量机：

```
ensembleLib.addModel("weka.classifiers.functions.SMO");
```

- AdaBoost，本身就是集成方法：

```
ensembleLib.addModel("weka.classifiers.meta.AdaBoostM1");
```

- LogitBoost，基于逻辑回归的集成方法：

```
ensembleLib.addModel("weka.classifiers.meta.LogitBoost");
```

- 决策树桩（Decision stump），基于单层决策树的集成方法：

```
ensembleLib.addModel("classifiers.trees.DecisionStump");
```

由于`EnsembleLibrary`的实现主要针对GUI与控制台用户，所以必须调用`saveLibrary(File,EnsembleLibrary, JComponent)`方法将模型存储到一个文件，代码如下：

```
EnsembleLibrary.saveLibrary(new
  File(path+"ensembleLib.model.xml"), ensembleLib, null);
System.out.println(ensembleLib.getModels());
```

接着实例化weka. classifiers.meta.EnsembleSelection类，对集成选择算法进行初始化。先了解方法的各个选项，如下所示。

- -L </path/to/modelLibrary>：指定modelLibrary文件，延伸所有模型列表。
- -W </path/to/working/directory>：指定存储所有模型的工作目录。
- -B <numModelBags>：设置包（bag）数目，即执行集成选择算法的迭代次数。
- -E <modelRatio>：设置库中模型的比例，这些模型会被随机选择迁移到模型的每个包。
- -V <validationRatio>：设置保留用于验证的训练数据集的比例。
- -H <hillClimbIterations>：设置每个模型包上要执行的爬山迭代次数。
- -I <sortInitialization>：设置集成库的比例，分类初始化算法将从该集成库选出，同时为每个模型包初始化集成方法。
- -X <numFolds>：设置交叉验证的折数。
- -P <hillclimbMettric>：指定爬山算法期间用作模型选择的指标，有效指标有accuracy、rmse、roc、precision、recall、fscore等。
- -A <algorithm>：指定集成选择使用的算法，有效算法有forward（默认，前进选择法）、backward（后退消去法）、both（前进与后退消去法）、best（简单打印集成库中的顶部执行者）、library（只训练集成库中的模型）。
- -R：该标记指示集成方法可否多次选择模型。
- -G：该项指定性能降低时分类初始化是否停止添加模型。
- -O：该选项用于详细输出。打印所有选中的模型性能。
- -S<num>：设置随机数种子值（默认1）。
- -D：开启该选项后，分类器将在调试模式下运行，并可能向控制台输出额外信息。

使用如下初始参数对算法进行初始化，并指定优化ROC指标：

```
EnsembleSelection ensambleSel = new EnsembleSelection();
ensambleSel.setOptions(new String[]{
  "-L", path+"ensembleLib.model.xml", // 模型库路径
    "-W", path+"esTmp", // 工作目录路径
"-B", "10", // 模型包数
  "-E", "1.0", // 模型比率.
  "-V", "0.25", // 验证比率
  "-H", "100", // 爬山迭代次数
"-I", "1.0", // 分类初始化
  "-X", "2", // 折数
  "-P", "roc", // 爬山指标
  "-A", "forward", // 算法
  "-R", "true", // 允许多次选择
  "-G", "true", // 性能下降时停止添加模型
  "-O", "true", // 输出详细信息.
```

```
    "-S", "1", // 随机数种子.
    "-D", "true" // 在调试模型下运行
  });
```

### 4.4.5　性能评估

做性能评估要耗费计算机大量计算力与内存，因此要保证使用额外的堆空间（比如**java–Xmx16g**）对JVM做初始化。即便如此，整个计算可能也要耗费几小时或者几天，具体取决于添加到模型库的算法数量。我们的示例将耗费4小时22分（12-core Intel Xeon E5-2420 CPU，32 GB内存），平均需要占用10% CPU与6 GB内存。

调用评估方法，输出如下结果：

```
double resES[] = evaluate(ensambleSel);
System.out.println("Ensemble Selection\n"
+ "\tchurn:     " + resES[0] + "\n"
+ "\tappetency: " + resES[1] + "\n"
+ "\tup-sell:   " + resES[2] + "\n"
+ "\toverall:   " + resES[3] + "\n");
```

模型库中的一组特定分类器得到如下结果：

```
Ensamble
  churn:     0.7109874158176481
  appetency: 0.786325687118347
  up-sell:   0.8521363243575182
  overall:   0.7831498090978378
```

总体来说，相比于本章开始时我们设计的初始参考基准，这个方法带来了明显的性能提升，提升幅度超过15个百分点。然而很难据此得出一个确切的结论，因为性能提升主要由3个因素引起：数据预处理与属性选择、多种学习方法的探索以及集成创建技术（该技术可以充分利用各种基本分类器而不会过度拟合）的应用。然而，性能提升意味着处理时间有明显增加，占用的内存也会增加。

## 4.5　小结

本章尝试解决了KDD Cup 2009挑战赛中提出的问题，即预测客户关系。解决问题的过程中，我们按照数据预处理步骤对数据做了预处理，处理了缺失值与多余属性。解决问题时，我们借用了KDD Cup挑战赛中获胜的解决方案，研究了如何使用各种学习算法应用集成方法，从而让分类性能得到显著提升。

下一章将尝试解决另外一个问题，即分析顾客的购买行为。这个过程中，我们将学习如何使用某些算法探查频繁出现的行为模式。

第 5 章

# 关联分析

关联分析（Affinity Analysis）是**购物篮分析**（**Market basket analysis，MBA**）的核心所在。通过关联分析，可以从特定用户或群组所做的活动中发现共现关系（co-occurrence relationships）。零售业中，关联分析能够帮助我们理解顾客的购买行为。在这些分析结果的帮助下，通过运用“聪明”的交叉销售（cross-selling）与追加销售策略（upselling strategies）可以大大提高商品销售额，而且能帮你制订忠诚度计划、商品促销与打折销售计划。



本章将学习如下主题：

- 购物篮分析
- 关联性规则学习
- 在各种领域的不同应用

首先了解与关联性规则学习有关的概念与算法，比如支持度（support）、提升度（lift）、**Apriori 算法**、**FP-增长算法**。然后使用Weka对超市数据集做关联分析，研究如何解释结果规则。最后分析如何将关联规则学习应用于其他领域，比如**IT运营分析**、医药等。

## 5.1　购物篮分析

随着大量电子销售点的设立，零售商们收集了数量惊人的数据。为了充分利用这些数据并使其产生商业价值，他们必须首先开发一种方法合并这些数据，以便了解业务的基本状况——他们在销售什么？多少销售点在运转？销售量是多少？

近来，人们将关注焦点转移到最低粒度级的购物篮交易上。在这个细节层次，零售商可以直接查看在他们店铺购物的每位顾客的购物篮，不仅可以知道顾客购买的商品数量，还可以了解顾客是如何搭配购买这些商品的。这些分析结果可以用于决定如何区分库存分类与指定商品，以及将同类别或不同类别的多种商品有效组合，以促进销售，获取更高利润。这些决策可以在整条零售链上实现，包括各种销售渠道、本地店铺，甚至针对某个特定顾客。这称为个性化营销，即针对每个顾客提供不同的商品。




图5-1　购物车问题

MBA涵盖的分析多种多样，如下所示。

- **商品关联**：定义同时购买两个或多个商品的可能性。
- **驱动商品的识别**：启动那些驱使人们去商店购买的商品识别，这些商品总是需要有货。
- **购物出行分类**：分析购物篮中的商品并对购物出行分类，比如每周购物出行、特殊情况等。
- **店到店对比**：了解购物篮数量，允许任何可被购物篮总数相除的度量指标，有效创建一个方便易用的方法，用于比较具有不同特征的商店（每个顾客销售量、每个交易收益、每个篮子的商品数等）。
- **收益优化**：帮助确定该店的神奇价格（magic price points），增加购物篮的大小与价值。
- **营销**：帮助找出可以产生更大利润的广告与促销方式，更精准地锁定报价以提高ROI（投资回报率），利用纵向分析产生更好的会员卡促销效果，吸引更多顾客进店购物。
- **经营优化**：通过商店定制、贸易区分类、优化商店布局，更好地匹配顾客需求与库存。

预测模型有助于零售商将合适的商品推荐给相应的客户群体，并帮助其了解顾客喜欢什么样的商品，预测顾客响应这个购物提议的可能性分数，了解顾客从这个提议的获益程度。

## 关联分析

关联分析用于确定顾客同时购买一组商品的可能性。零售业中，商品之间存在着一些天然的

关联，比如一个很常见的例子是：买汉堡肉饼的人通常还会一起买汉堡面包、番茄酱、芥末酱、番茄，以及其他做汉堡包的原料。

有些商品的关联看起来可能微不足道，有些关联则不是很明显。一个典型的例子是牙膏和金枪鱼。看起来吃金枪鱼的人吃完饭后更倾向于立刻刷牙。那么，对于零售商来说，为什么把握商品之间的关联显得如此重要？这些信息对于策划促销活动至关重要，对一些商品做降价促销可能引起相关商品销售额激增，而对于这些相关商品则无需专门再做促销活动。

接下来将学习关联规则学习算法：Apriori算法与FP-增长算法。

## 5.2　关联规则学习

关联规则学习是一种很流行的方法，用于在大型数据库中发现两个项目之间的有趣联系。它通常应用于零售业，揭示商品之间的关联规则。

关联规则学习方法使用不同的兴趣度度量方法，以发现数据库中那些有趣的强规则模式。比如，下面规则表明，如果一个顾客同时购买了洋葱与土豆，那么他很可能会买汉堡包肉饼：{洋葱, 土豆} → {汉堡包}。

另一个典型的案例是啤酒与尿布的故事，你可能已经在机器学习有关课程中听过。一项超市顾客购物行为分析指出：购买尿布的顾客（大概是年轻男士）往往也会购买啤酒。这立刻成为一个流行的例子，用来说明如何从每日数据中找到意想不到的关联规则。然而，对于这个故事的真实性，人们持有不同看法。丹尼尔·鲍尔斯说（DSS新闻，2002）：

> “1992年，Thomas Blischok（天睿公司零售业顾问团经理）和他的团队成员打算对120万个购物篮数据进行分析，这些数据来自大约25家Osco Drug商店。他们对数据库查询做了改动，使之可以对商品之间的关联进行识别。分析结果指出：下午5:00~7:00的确发现消费者同时购买了啤酒与尿布。但Osco的经理们没有利用啤酒与尿布之间的关系，并且也没有把它们摆放在一起。”

除了上面这个来自MBA的例子之外，现在关联规则也被应用于许多领域，包括网络使用挖掘、入侵检测、连续生产、生物信息学。本章后半部分将详细了解这些内容。

### 5.2.1　基本概念

开始学习算法之前，先了解基本概念。

#### 1. 交易数据库

关联规则挖掘中，数据集的结构与第1章使用的那些数据集的结构略有不同。首先，数据集

5

中没有类值（class value），学习关联规则时不需要它。其次，数据集表现为一个交易表单，每个超市商品对应于一个二元属性。因此，特征向量可能会非常大。

请思考下面这个例子。假设我们有4张收据，如图5-2所示。每个收据对应于一笔购物交易。

WWW.MACHINE-LEARNING-JAVA.COM
GROCERY STORE
921 JAVA AVENUE
NEW YORK
NY
9999
PURCHASE:
POTATEOS $4.12
BURGER $12.04
VAT +11% TAX: $1.77
TOTAL: $17.93
PAYMENT METHOD: CREDIT CARD
TRANSACTION #1458293367 -001
DATE:10/03/2016 9:29:27 AM
THANK YOU

WWW.MACHINE-LEARNING-JAVA.COM
GROCERY STORE
921 JAVA AVENUE
NEW YORK
NY
9999
PURCHASE:
POTATEOS $4.12
BURGER $12.04
ONIONS $3.14
BEER $27.55
VAT +11% TAX: $5.15
TOTAL: $52.00
PAYMENT METHOD: CREDIT CARD
TRANSACTION #1458293420 -001
DATE:10/03/2016 9:30:20 AM
THANK YOU

WWW.MACHINE-LEARNING-JAVA.COM
GROCERY STORE
921 JAVA AVENUE
NEW YORK
NY
9999
PURCHASE:
DIPPERS $29.95
BEER $27.55
VAT +11% TAX: $6.33
TOTAL: $63.83
PAYMENT METHOD: CREDIT CARD
TRANSACTION #1458293508 -001
DATE:10/03/2016 9:31:40 AM
THANK YOU

WWW.MACHINE-LEARNING-JAVA.COM
GROCERY STORE
921 JAVA AVENUE
NEW YORK
NY
9999
PURCHASE:
BURGER $12.04
ONIONS $3.14
BEER $27.55
VAT +11% TAX: $4.70
TOTAL: $47.43
PAYMENT METHOD: CREDIT CARD
TRANSACTION #1458293459 -001
DATE:10/03/2016 9:30:59 AM
THANK YOU

图5-2　购物交易收据

为了以交易数据库的形式写出这些收据，首先要识别出现在收据上的所有商品，这些商品有洋葱、土豆、汉堡包、啤酒与勺子。每一次购物（交易）写在一行中，若交易中购买了某个商品，就将其标为1，否则标为0，如表5-1所示。

**表5-1　交易商品列表**

| 交易ID | 洋葱 | 土豆 | 汉堡包 | 啤酒 | 勺子 |
|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 | 0 |
| 2 | 1 | 1 | 1 | 1 | 0 |
| 3 | 0 | 0 | 0 | 1 | 1 |
| 4 | 1 | 0 | 1 | 1 | 0 |

这个示例包含的交易很少，只有4个。实际应用中，数据集通常包含几千或者几百万笔交易，这使得学习算法可以从这些交易中发现显著的统计模式。

### 2. 项目集与规则

项目集只是一组项目的集合，比如{洋葱, 土豆, 汉堡包}。规则由两个项目集$X$与$Y$组成，格式为：$X \rightarrow Y$。

这表示一种模式，即观察项目集$X$时，也会观察项目集$Y$。为了选择有趣的规则，你可以使用各种有意义的度量方法。

### 3. 支持度

对某个项目集而言，支持度指包含该项目集的交易在总交易中所占的比例。比如前面表格中

的{土豆, 汉堡包}项目集，它的支持度是50%，即它在50%的交易中出现。总交易数为4，包含{土豆, 汉堡包}项目集的交易有两笔，所以supp({土豆, 汉堡包}) = 2/4 = 0.5。

直观地说，支持度表示总交易中有多少笔交易支持这个模式。

#### 4. 置信度

一个规则的置信度指其准确度，其定义如下：

$$\mathrm{Conf}(X \rightarrow Y)=\mathrm{supp}(X \cup Y)/\mathrm{supp}(X)$$

比如，前面表格中，{洋葱, 汉堡包}→{啤酒}规则的置信度为0.5/0.5=1.0，100%表示购买洋葱与汉堡包的顾客也会同时购买啤酒。

### 5.2.2 Apriori 算法

Apriori算法十分经典，用于频繁模式挖掘与基于事务的关联规则学习。通过在数据库中找出单独的频繁项，并将之扩展到更大的项集，Apriori算法能够产生关联规则，突出数据库的总体趋势。



Apriori算法先创建一组项目集，比如项目集1={项目A, 项目B}，并且计算支持度，即计算它在数据库中出现的次数。然后Apriori算法使用自下而上的方法对频繁项集做扩展，一次一项。它工作时会消去最大集合，把更小一点的集合作为候选，并且认识到一个大集合不可能是频繁的，除非它的所有子集都是频繁集。找不到更多扩展时，算法即停止。

虽然Apriori算法是机器学习中的一个重要里程碑，但它仍然存在一些效率不佳与权衡取舍的问题。接下来学习更新的FP-增长算法。

### 5.2.3 FP-增长算法

FP-增长（**FP，频繁模式**）算法中，将交易数据库表示为一个前缀树。首先，在算法数据集中计算数据项出现的次数。第二步中，算法创建前缀树，它是一个有序树数据结构，通常用于存储字符串。使用前缀树表示前面例子，如图5-3所示。

如果许多事务共享大部分频繁项，那么前缀树就会产生很高的压缩效果，几乎压到树根。大项目集直接增长，不会产生候选项，也不会对整个数据库测试它们。增长从树底部开始，通过匹配最小支持度与置信度查找所有项目集。一旦递归过程结束，所有带有最小覆盖率的大项目集就会被找到，并开始创建关联规则。

FP-增长算法有几个优点：首先，它创建一个FP树，使用非常紧凑的表示形式对原数据集进行编码；其次，它能够有效创建频繁项目集，并且可以充分利用**FP树结构**与**分而治之**的策略。

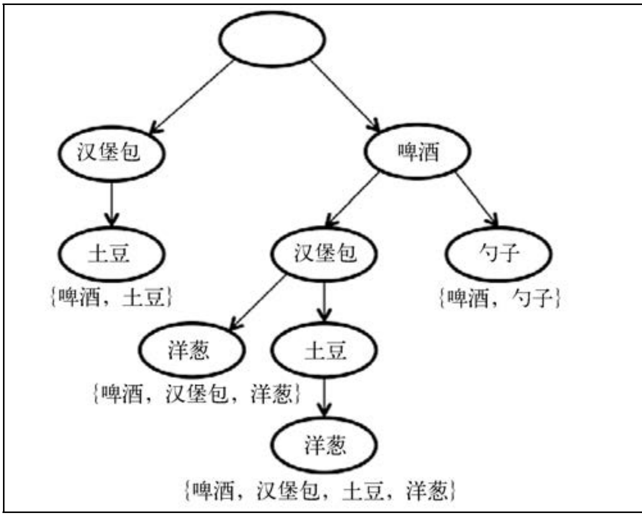



图5-3　交易商品前缀树

## 5.2.4　超市数据集

超市数据集位于`datasets/chap5/supermarket.arff`，它描述了超市顾客的购物习惯。大部分属性表示一个特定的商品组，比如乳制品、牛肉、土豆；或者部门，比如部门79、部门81等。下图显示了数据库的一个片段，如果顾客购买了某件商品，就将其标为1，否则标为0。其中，每一个客户就是一个实例。数据集不包含类属性（class attribute），学习关联规则时并不需要它。数据样本如表5-2所示。

表5-2　超市数据集样本

| coffee | sauces-gravy-pkle | confectionary | puddings-deserts | dishcloths-scour | deod-disinfectan1 | frozen foods | razor blades | fuels-garden aids | spices | jams-spreads |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |

## 5.3 发现模式

下面，使用前面学过的两种算法（Apriori算法与FP-增长算法）发现购物模式。

### 5.3.1 Apriori 算法

首先，使用已经在Weka中实现的Apriori算法。该算法不断减小最小支持度，直到找到所需的规则数，它带有给定的最小置信度。

```
import java.io.BufferedReader;
import java.io.FileReader;
import weka.core.Instances;
import weka.associations.Apriori;
```

其次，加载超市数据集，代码如下：

```
Instances data = new Instances(
new BufferedReader(
new FileReader("datasets/chap5/supermarket.arff")));
```

接着，初始化Apriori实例，调用`buildAssociations(Instances)`函数，开始频繁模式挖掘，代码如下：

```
Apriori model = new Apriori();
model.buildAssociations(data);
```

最后，输出发现的项目集与规则，代码如下：

```
System.out.println(model);
```

输出结果如下：

```
Apriori
=======

Minimum support: 0.15 (694 instances)
Minimum metric <confidence>: 0.9
Number of cycles performed: 17

Generated sets of large itemsets:
Size of set of large itemsets L(1): 44
Size of set of large itemsets L(2): 380
Size of set of large itemsets L(3): 910
Size of set of large itemsets L(4): 633
Size of set of large itemsets L(5): 105
Size of set of large itemsets L(6): 1

Best rules found:

 1. biscuits=t frozen foods=t fruit=t total=high 788 ==> bread and cake=t 723
```

```
<conf:(0.92)> lift:(1.27) lev:(0.03) [155] conv:(3.35)
 2. baking needs=t biscuits=t fruit=t total=high 760 ==> bread and cake=t 696
<conf:(0.92)> lift:(1.27) lev:(0.03) [149] conv:(3.28)
 3. baking needs=t frozen foods=t fruit=t total=high 770 ==> bread and cake=t 705
<conf:(0.92)> lift:(1.27) lev:(0.03) [150] conv:(3.27)
...
```

这个算法根据置信度输出10个最佳规则。先看第一个规则，并对输出进行解释，如下所示：

```
biscuits=t frozen foods=t fruit=t total=high 788 ==> bread and cake=t 723
<conf:(0.92)> lift:(1.27) lev:(0.03) [155] conv:(3.35)
```

上述规则表明：顾客同时购买饼干、冷冻食品与水果且总价较高时，很有可能也会购买面包与蛋糕。{饼干，冷冻食品，水果，总购买价格高}项目集出现在788笔交易中，而{面包，蛋糕}项目集出现在723笔交易中。这个规则的置信度为0.92，表示其在92%的交易中都适用，即出现{饼干，冷冻食品，水果，总购买价格高}项目集。

输出结果也显示了其他指标，比如提升度（lift）、杠杆率（leverage）、确信度（conviction），用于评估相对于最初假设的准确程度。比如确信度为3.35，表示在关联完全随机的情形下，规则不正确的可能性将是平常的3.35倍。提升度表示$X$与$Y$相比于预期同时出现的次数，前提是它们是统计独立的（`lift=1`）。$X \rightarrow Y$规则中，提升度为2.16表示$X$的几率比$Y$的几率大2.16倍。

## 5.3.2　FP-增长算法

下面尝试使用更有效率的FP-增长算法得到一样的结果。FP-增长算法也是在`weka.associations`包中实现的。

```
import weka.associations.FPGrowth;
```

就像我们前面所做的那样，首先对FP-增长算法进行初始化。

```
FPGrowth fpgModel = new FPGrowth();
fpgModel.buildAssociations(data);
System.out.println(fpgModel);
```

输出结果表明FP-增长算法发现了16个规则：

```
FPGrowth found 16 rules (displaying top 10)

 1. [fruit=t, frozen foods=t, biscuits=t, total=high]: 788 ==> [bread and cake=t]: 723
<conf:(0.92)> lift:(1.27) lev:(0.03) conv:(3.35)
 2. [fruit=t, baking needs=t, biscuits=t, total=high]: 760 ==> [bread and cake=t]: 696
<conf:(0.92)> lift:(1.27) lev:(0.03) conv:(3.28)
...
```

可以看到，用FP-增长算法找到的规则与Apriori算法差不多是一样的。但处理大型数据集时，使用FP-增长算法需要的时间明显更短，执行效率更高。

## 5.4　在其他领域中的应用

上面介绍了关联分析在揭示超市顾客购物行为模式方面的应用。尽管关联规则学习根植于对销售点的交易分析，但它也可以应用于零售业之外的其他领域，用来在其他类型的“购物篮”中寻找关联。“购物篮”这个概念可以很容易地扩展到服务与产品，比如分析使用信用卡购买的服务，像租车、租房；又比如分析与电信用户购买的增值服务相关的信息（呼叫等待、呼叫转移、DSL、缩位拨号等），这可以帮助运营商决定如何改进他们提供的服务包。

接下来，了解如下这些跨行业应用的例子：

- ❑ 医疗诊断
- ❑ 蛋白质序列
- ❑ 人口普查
- ❑ 客户关系管理
- ❑ IT运营分析



### 5.4.1　医疗诊断

医疗诊断中，关联规则可以辅助医生对患者进行治疗。诊疗过程中一个常见的难题是，很难归纳出可靠的诊断规则。从理论上说，归纳过程不能完全保证归纳假设本身的正确性。事实上，诊断过程本身不是一个轻松的事，因为它会涉及不可靠的诊断测试，而且训练样本中可能还存在噪声问题。

尽管如此，我们仍然可以使用关联规则识别可能同时出现的症状。这种应用情景中，一个事务对应于一个医疗案例，而症状对应于项目。治疗病人时会记录一系列症状，它们一起组成一个事务。

### 5.4.2　蛋白质序列

有关蛋白质的大量研究已经深入到了解其组成与性质中来，然而还有许多方面有待人们认真研究。现在普遍的共识是：蛋白质的氨基酸序列不是随机的。

借助关联规则，我们有可能识别同一个蛋白质中不同氨基酸之间的关联性。蛋白质序列由20种氨基酸组成。每种蛋白质拥有独一无二的三维结构，具体取决于氨基酸序列，序列中微小的改变都会导致蛋白质功能发生变化。应用关联规则时，一种蛋白质对应于一个事务，氨基酸与其结构对应于项目。

这些关联规则有助于加深我们对蛋白质组成的理解，有可能帮助我们在蛋白质的一些特殊的氨基酸组中找到有关全球相互作用的线索。这些关联规则或约束的知识对于人工合成蛋白质非常有用。

### 5.4.3 人口普查数据

人口普查产生大量与社会有关的统计信息，这些信息既可以提供给研究者使用，也可以提供给普通大众。有关人口数量与经济普查的信息可以用在公共服务（教育、医疗、交通、基金）与商业（设立新工厂、购物中心、银行甚至推销特定商品）规划中。

为了发现频繁模式，每个统计区域（比如自治区、城市、邻区）对应于一个事务，收集的指标对应于项目。

### 5.4.4 客户关系管理

第4章已经简单讨论过**客户关系管理**（**CRM**），它是一个富数据源，公司希望通过它找出不同客户组对产品、服务的偏好，借此加强产品、服务、客户之间的黏合度。

关联规则能够强化知识管理流程，帮助营销人员更好地了解他们的客户，以便向客户提供更优质的服务。比如，关联规则可以从客户资料与销售数据中检测客户在不同时间段行为的变化。基本思想是从两个数据集找到变化，并从每个数据集产生规则进行规则匹配。

### 5.4.5 IT 运营分析

在大量事务记录基础之上，关联规则学习很适合用于分析每天IT运营定期收集的数据，启用IT运营分析工具检测频繁模式并找到主要变化。IT专家需要查看总体运营情况，并了解某个问题带来的影响，比如一个数据库的问题如何影响应用程序服务器。

在一个特殊的日子，IT运营可能发现许多警报，并进一步发现它们来自一个事务数据库。此情形下，使用关联规则学习算法后，IT运营分析工具可以检测到同时出现这些警报的频繁模式。这能帮助我们更好地了解组件之间是如何相互影响的。

借助发现的这些警告模式，可以进行预测分析。比如某个特定的数据库服务器中包含一个Web应用程序，突然触发一个数据库警报。通过查看关联规则学习算法发现的频繁模式，IT工作人员可以采取相应措施，避免问题进一步影响Web应用程序。

关联规则学习也能发现那些源自相同IT事件的警报事件。比如每次添加新用户时，Windows操作系统中就会检测到6个变化。接着，在**应用组合管理工具**（**Application Portfolio Management，APM**）中，IT可能面临多个警报，显示数据库中的事务时间为“高”。如果所有这些问题都来自同一个源（比如几百个更改警报都是由Windows更新引起的），那么这个频繁模式挖掘可以帮助我们快速删减警报数量，让IT运维人员专注于最关键的改变。

## 5.5　小结

本章学习了应用事务数据集上的关联规则学习以深入了解频繁模式的方法，并且在Weka中做了关联分析演示。我们从中认识到，最困难的部分在于对结果的分析，解释规则时要小心留意，因为关联（即相关性）与因果关系是不同的。

下一章将学习如何使用Apache Mahout库解决商品推荐问题。Apache Mahout是一个可扩展的机器学习库，它可以处理大数据。

第 6 章

# 使用Apache Mahout制作推荐引擎

6

推荐引擎可能是当今初创公司应用最多的一种数据科学方法。有两项主要技术用来创建一个推荐系统：**基于内容的过滤与协同过滤**。基于内容的过滤算法使用项目属性寻找带有相似属性的项目；协同过滤算法关注的是用户的评分或者其他用户的行为，它基于拥有类似行为的用户喜好与购买的物品进行推荐。

本章先讲解基本概念，它们是理解推荐引擎原理必需的内容；然后演示如何使用Apache Mahout中的各种算法实现快速创建一个可扩展的推荐引擎。本章内容涵盖如下主题：

- 如何创建一个推荐引擎
- 准备Apache Mahout
- 基于内容的方法
- 协同过滤方法

到本章结束时，你将知道我们的问题适合使用哪种推荐引擎进行解决，以及如何快速创建这样一个推荐引擎。

## 6.1 基本概念

推荐引擎的目标是向用户展示他们感兴趣的项目。与搜索引擎不同的是，相关内容通常出现在一个网站中，用户不必创建查询来请求它。因为推荐引擎会观察用户的行为，并且在用户不知情的情况下为他们创建查询。

可以这么说，推荐引擎最著名的例子就是www.amazon.com，它使用多种方法为用户做个性化推荐。图6-1向我们展示了一种商品推荐的例子——“**购买这件商品的顾客还买了……**”。这是一个基于项目的协同推荐的例子，在这种推荐方式下，与特定项目相似的项目都会得到推荐。相关内容稍后讲解。




图6-1　一个推荐引擎的例子——www.amazon.com

本节将介绍几个关键概念，它们与理解和创建推荐引擎息息相关。

## 6.1.1　关键概念

推荐引擎需要如下4个输入做出推荐：

- 使用属性描述的项目信息；
- 用户资料，比如年龄范围、性别、位置、朋友等；
- 用户交互，比如评级、浏览、标记、比较、保存、电邮；
- 显示项目上下文，比如项目的分类与地理位置。

获得这些输入后，推荐引擎将其组合在一起，帮助我们回答如下问题：

- 购买、观看、浏览、收藏过这个项目的用户还买了、看了、浏览了、收藏了……
- 与这个项目类似的项目
- 你可能认识的其他用户
- 和你类似的其他用户

下面详细介绍这种组合是如何工作的。

## 6.1.2　基于用户与基于项目的分析

创建推荐引擎时要搞清楚，推荐引擎尝试推荐一个特定项目时，搜索的是相关项目还是相关用户。

基于项目的分析中，引擎的主要任务是找出那些与特定项目类似的项目；而基于用户的分析中，引擎首先要找出那些与特定用户类似的用户。比如，先找出那些带有相同资料信息（年龄、性别等）或行为历史（买了、看了、浏览了等）的用户，然后将相同项目推荐给其他类似用户。

这两种方法都要求计算一个相似矩阵（similarity matrix），具体取决于分析的是项目属性还是用户行为。下面深入了解具体应该如何做。

### 6.1.3　计算相似度的方法

计算相似度（similarity）的基本方法有3种：

- 协同过滤算法关注的是用户评分或其他用户的行为，并且基于拥有类似行为的用户喜好与购买的物品进行推荐；
- 基于内容的过滤算法使用项目属性寻找带有相似属性的项目；
- 组合了协同过滤与基于内容的过滤的混合方法。

接下来，详细学习每一种方法。

#### 1. 协同过滤

协同过滤只基于用户评级或其他用户的行为，基于拥有相似行为的用户喜好与购买的物品进行推荐。

协同过滤的主要优点是不依赖于项目内容，因此可以准确推荐复杂项目，而无需了解项目本身，比如电影。它基于的假设是“人们过去认可的将来也会认可”“他们喜欢与过去喜欢的项目相似的项目”。

这个方法的主要缺点就是所谓的冷启动（cold start），也就是说，如果想创建一个精确的协同过滤系统，算法往往需要先有大量用户评分。因此，产品的第一个版本中通常不会使用协同过滤，直到有了相当数量的数据积累之后才会使用。

#### 2. 基于内容的过滤

另一方面，基于内容的过滤建立在项目的描述与用户偏好资料之上，按照如下步骤进行组合：首先，使用属性描述项目并找出相似项目，我们选用一个距离测度（比如**余弦距离**或**皮尔逊相关系数**，详细内容请参考第1章有关距离测度的内容）测量项目之间的距离。接着，将用户资料输入方程式。鉴于用户喜欢的项目类型的反馈，我们引入权重指示特定项目属性的重要程度。比如，“潘多拉电台流媒体服务应用”基于内容的过滤技术使用400多个属性创建电台。起初，一个用户通过特定属性挑选了一支歌曲，并通过提供反馈突出重要的歌曲属性。

这个方法最初只需要很少的用户反馈信息，因此它能有效避免冷启动问题。

#### 3. 混合方法

那么，应该如何选择协同过滤方法与基于内容的过滤方法呢？借助协同过滤方法可以从用户对一个内容源的行为了解用户喜好，并通过用户偏好找出其他类型的内容。基于内容的过滤方法仅限于推荐同类型的内容，并且用户已经在使用这种类型。这对于不同的使用案例是有价值的，比如基于用户正在浏览的新闻推荐新闻文章是有用的。但如果能够再进一步，基于正在浏览的新闻推荐其他类型的资源（比如图书、电影），这将会更有用。

协同过滤与基于内容的过滤不是相互排斥的，某些情况下，我们可以将二者组合以产生更有效的结果。比如，**Netflix**使用协同过滤分析相似用户的搜索与观看模式。此外，还使用基于内容的过滤向用户推荐高评分影片。

混合技术有很多，比如加权混合、切换混合、分区混合、特征组合、特征扩充、级联混合、分层混合等。机器学习与数据挖掘社区中，推荐系统一直是个活跃版块，数据科学会议上也会专门为它设立分会场。通过Adomavicius与Tuzhilin（2005）的Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions论文，你可以很好地了解这些技术。论文中，作者讨论了不同方法与基本算法，并且提供了更多有价值的参考论文。需要认真了解某个特定方法时，为了获取更多技术细节，你可以阅读弗朗西斯科·里奇等人合著的《推荐系统：技术、评估及高效算法》。



### 6.1.4　利用与探索

推荐系统中，不同推荐项目之间总是存在权衡取舍，一类推荐项目会落入用户甜区，它基于我们对用户已有的了解（**利用**）；另一类推荐项目不会陷入用户甜区，其目标是向用户展示一些新奇物品（**探索**）。带有一点探索特征的推荐系统只会推荐与前一个用户评价一致的项目，不会显示那些不在当前范围的项目。事实上，我们也经常需要从用户甜区外收获新项目，这通常会带来惊喜，也有助于发掘用户新的甜区。

本节讨论了创建推荐引擎必需的概念，接下来学习如何使用Apache Mahout实际创建一个推荐引擎。

## 6.2　获取 Apache Mahout

第2章已经介绍过Mahout，它是一个可扩展的机器学习库，提供了丰富的组件。借助这些组件，你可以使用所选算法创建一个定制的推荐系统。Mahout的创造者说，它是一个企业级的、性能良好的、具有高可扩展性与灵活性的机器学习库。

可以使用两种方法配置并运行Mahout：一种是单机运行（有无Hadoop均可）；另一种是分布式处理。我们把重点放在没有配置Hadoop的Mahout上。关于Mahout更多高级配置与使用方法，推荐你阅读如下*Learning Apache Mahout*和*Learning Apache Mahout Classification*。

由于Apache Mahout的创建与发布系统都是基于Maven的，所以需要先学习如何安装。下面介绍一种最简单的安装方法，需要用到带有Maven插件的Eclipse。

## 在带有 Maven 插件的 Eclipse 中配置 Mahout

需要使用最新版本的Eclipse，可以从Eclipse官方站点下载。本书选用Eclipse Luna。打开Eclipse，新建一个**Maven**项目，保持默认设置不变，如图6-2所示。

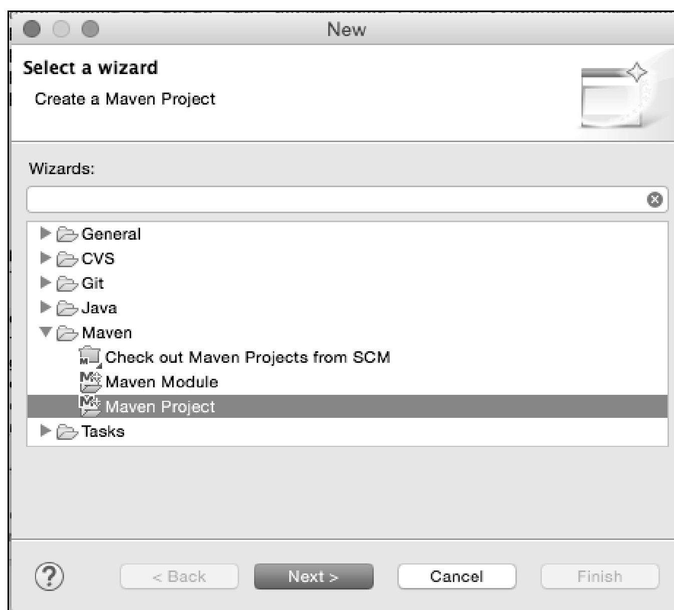



图6-2　新建项目页面

单击Next按钮，弹出**New Maven project**窗口，如图6-3所示。

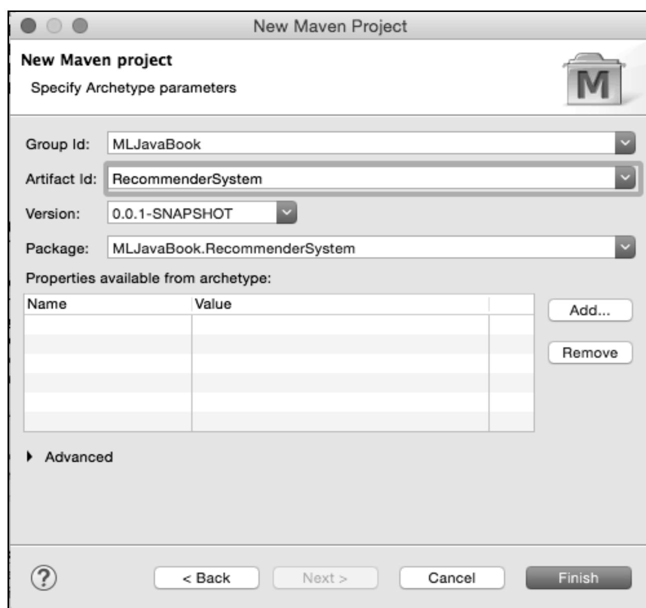



图6-3　New Maven project窗口

接下来，要向项目添加Mahout jar文件及其依赖包。找到pom.xml文件，使用文本编辑器打开（点击pom.xml后依次选择**Open with>Text Editor**），如图6-4所示。



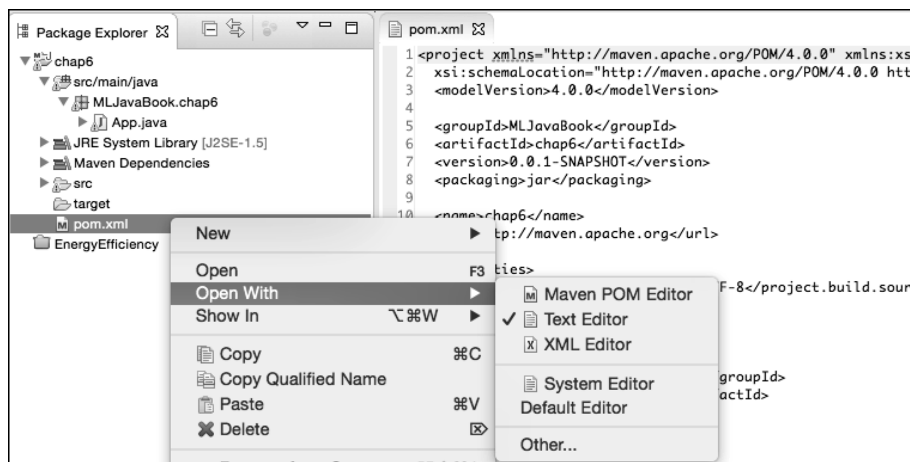



图6-4　使用文本编辑器打开pom.xml文件

在pom.xml文件中找到以`<dependencies>`开始的行，紧接着添加如下代码：

```
<dependency>
  <groupId>org.apache.mahout</groupId>
```

```
  <artifactId>mahout-mr</artifactId>
  <version>0.10.0</version>
</dependency>
```

这样就添加好了Mahout库，现在可以正常使用了。

## 6.3　创建一个推荐引擎

为了演示基于内容的过滤与协同过滤方法，下面创建一个图书推荐引擎。

### 6.3.1　图书评分数据集

本章将使用图书评分数据集（Ziegler等，2005），它由一个爬虫程序收集了4周而得到。该数据集包含的数据涉及**Book-Crossing**网站的278 858个会员与1 157 112个评分，这些评分既有隐含的也有明确的，涵盖271 379个不同的ISBN。用户数据经过匿名化处理，但含有人口统计信息。你可以从如下网站获得该数据集：

http://www2.informatik.uni-freiburg.de/~cziegler/BX/

Book-Crossing数据集包含3个文件，网站中对于这3个文件的描述如下。

- **BX-Users**：包含用户。请注意，用户ID（User-ID）被匿名化处理，由字符串换为整数。如果存在统计数据则给出（位置与年龄），否则这些字段就是NULL值。
- **BX-Books**：图书通过各自不同的ISBN号码加以识别。无效ISBN已经被从数据集中移走。而且，给出了一些基于内容的信息（图书名称、图书作者、出版年份、出版商），这些信息是从Amazon Web Service那里得到的。请注意，如果图书有多个作者，则只提供第一作者。此外还给出了指向封面图片的URL连接，有三种不同形式：Image-URL-S、Image-URL-M、Image-URL-L，即小、中、大。这些URL指向Amazon网站。
- **BX-Book-Ratings**：包含图书的评分信息。有些评分（Book-Rating）很明确，使用数字1~10表示（分值越高表示越值得阅读）；有些评分不明，使用0表示。

### 6.3.2　加载数据

根据数据的存储位置（文件或数据库），有两种不同的数据加载方法可以选用。首先详细讲解如何从文件加载数据，包括如何处理自定义格式。之后快速了解如何从数据库加载数据。

#### 1. 从文件加载数据

可以使用`FileDataModel`类从文件加载数据，文件中的数据以逗号进行分隔，每一行顺序包含`userID`、`itemID`、`preference`（可选）、`timestamp`（可选），格式如下：

```
userID, itemID[, preference[, timestamp]]
```

可选项preference是一个二元偏好值，也就是说，对于某本书，用户要么“喜欢”，要么“不喜欢”，没有喜爱程度的差别。

以#开始的行或空行都会被忽略。数据行也可以包含其他字段，但这些字段会被忽略。

DataModel类可以接受如下类型：

- userID、itemID是long类型
- preference是double类型
- timestamp是long类型

如果你能提供上面这种格式的数据集，那就可以简单地使用如下代码加载数据：

```
DataModel model = new FileDataModel(new File(path));
```

这个类不适合用于加载大量数据，比如几千万行数据。需要加载大量数据时，使用带有JDBC支持的DataModel与数据库会更合适。

现实情况下，我们无法保证提供给我们的输入数据中，userID与itemID总为整型值。比如，示例中，itemID对应于ISBN书号，它可以唯一地标识一本图书，那么就不是整型值。默认情况下，FileDataModel不适合处理这种数据。



下面考虑itemID是字符串时应该如何处理。我们要定义自己的数据模型，对FileDataModel做扩展，重载readItemIDFromString(String)方法，读入字符串形式的itemID值，而后将其转换为long型值并返回。为了将String转化为long，我们要对Mahout中的AbstractIDMigrator辅助类做扩展，这个类的设计初衷就是为了完成这个任务。

下面看看如何对FileDataModel做扩展：

```
class StringItemIdFileDataModel extends FileDataModel {

  // 初始化将String转换为long的转换器
  public ItemMemIDMigrator memIdMigtr;

  public StringItemIdFileDataModel(File dataFile, String regex)
    throws IOException {
    super(dataFile, regex);
  }

  @重载
  protected long readItemIDFromString(String value) {

    if (memIdMigtr == null) {
      memIdMigtr = new ItemMemIDMigrator();
    }
```

```
    // 转换为long
    long retValue = memIdMigtr.toLongID(value);
    // 存储到缓存
    if (null == memIdMigtr.toStringID(retValue)) {
      try {
        memIdMigtr.singleInit(value);
      } catch (TasteException e) {
        e.printStackTrace();
      }
    }
    return retValue;
  }

  // 将long转换为Sring
  String getItemIDAsString(long itemId) {
    return memIdMigtr.toStringID(itemId);
  }
}
```

对其他有用的方法进行重载，如下。

- `readUserIDFromString(String value)`：如果用户ID不是数字。
- `readTimestampFromString(String value)`：更改解析时间戳的方式。

接下来，对`AbstractIDMIgrator`类做扩展：

```
class ItemMemIDMigrator extends AbstractIDMigrator {

  private FastByIDMap<String> longToString;

  public ItemMemIDMigrator() {
    this.longToString = new FastByIDMap<String>(10000);
  }

  public void storeMapping(long longID, String stringID) {
    longToString.put(longID, stringID);
  }

  public void singleInit(String stringID) throws TasteException {
    storeMapping(toLongID(stringID), stringID);
  }

  public String toStringID(long longID) {
    return longToString.get(longID);
  }
}
```

以上就是所有准备工作。下面加载数据集，代码如下：

```
StringItemIdFileDataModel model = new StringItemIdFileDataModel(
  new File("datasets/chap6/BX-Book-Ratings.csv"), ";");
System.out.println(
```

```
"Total items: " + model.getNumItems() +
"\nTotal users: " +model.getNumUsers());
```

上述代码分别输出项目与用户总数，如下：

```
Total items: 340556
Total users: 105283
```

至此，从文件载入数据集的工作就完成了，接下来开始推荐。不过在此之前，还是先了解一下如何从数据库加载数据。

### 2. 从数据库加载数据

除了从文件加载数据之外，还可以从数据库中加载数据，这需要用到一个JDBC数据模型。本章不会详细讲解安装数据库、连接数据库等内容，只大致了解应该怎样做。

由于数据库连接器存在于一个单独的包——mahout-integration中，所以首先要把这个包添加到项目的依赖列表。打开pom.xml文件，添加如下依赖关系：

```
<dependency>
  <groupId>org.apache.mahout</groupId>
  <artifactId>mahout-integration</artifactId>
  <version>0.7</version>
</dependency>
```



由于要连接MySQL数据库，所以还需要向项目添加一个用于处理数据库连接的包。在pom.xml文件中添加如下代码：

```
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>5.1.35</version>
</dependency>
```

现在，我们已经把需要的所有包都准备好了，接下来创建连接。首先，使用连接详细信息初始化DataSource类，代码如下：

```
MysqlDataSource dbsource = new MysqlDataSource();
  dbsource.setUser("user");
  dbsource.setPassword("pass");
  dbsource.setServerName("hostname.com");
  dbsource.setDatabaseName("db");
```

Mahout集成了针对各种数据库的JDBCDataModel实现，使得我们可以通过JDBC访问这些数据库。默认情况下，这个类假定在JNDI名称jdbc/taste之下有DataSource可用，允许我们使用一个taste_preferences表访问数据库，格式如下：

```
CREATE TABLE taste_preferences (
  user_id BIGINT NOT NULL,
```

```
  item_id BIGINT NOT NULL,
  preference REAL NOT NULL,
  PRIMARY KEY (user_id, item_id)
)
CREATE INDEX taste_preferences_user_id_index ON taste_preferences
  (user_id);
CREATE INDEX taste_preferences_item_id_index ON taste_preferences
  (item_id);
```

对数据库支持（database-backed）的数据模式做如下初始化。除了DB连接对象之外，还可以指定自定义的表名与表列名，如下：

```
DataModel dataModel = new MySQLJDBCDataModel(dbsource,
  "taste_preferences",
  "user_id", "item_id", "preference", "timestamp");
```

3. 内存数据库

最后，数据模型也可以在内存中动态创建并保存。可以从一个用户偏好数组创建数据库，这个偏好数组保存着用户对一组项目的评分。

整个创建过程如下：首先，创建一个`FastByIdMap`散列表，它映射到存储一组用户偏好的数组`PreferenceArray`。

```
FastByIDMap <PreferenceArray> preferences = new FastByIDMap
  <PreferenceArray> ();
```

接下来，为用户新建一个偏好数组，存储用户评分。初始化这个数组时，必须给出占用内存大小的参数。

```
PreferenceArray prefsForUser1 =
  new GenericUserPreferenceArray (10);
```

接下来，为当前偏好（`0`号位置）设置用户ID。其实，这将为所有偏好设置用户ID。

```
prefsForUser1.setUserID (0, 1L);
```

为当前偏好（`0`号位置）设置项目ID。

```
prefsForUser1.setItemID (0, 101L);
```

为当前偏好（`0`号位置）设置偏好值。

```
prefsForUser1.setValue (0, 3.0f);
```

继续为用户设置其他项目：

```
prefsForUser1.setItemID (1, 102L);
prefsForUser1.setValue (1, 4.5F);
```

最后，添加用户偏好到散列映射：

```
preferences.put (1L, prefsForUser1); //userID用作键
```

接着，使用偏好散列映射初始化`GenericDataModel`：

```
DataModel dataModel = new GenericDataModel(preferences);
```

上述代码演示了如何为一个用户添加两个偏好，然而在实际应用中，可能需要为多个用户添加多个偏好。

## 6.3.3 协同过滤

可以使用Mahout中的`org.apache.mahout.cf.taste`包创建推荐引擎，这个包之前是一个名叫`Taste`的单独项目，现已并入Mahout中被继续开发。

基于Mahout的协同过滤引擎接收用户对某些项目的偏好（嗜好），返回用户可能喜欢的其他项目。比如，一个销售图书或CD的网站使用Mahout后，可以很轻松地根据顾客以往的购物数据找出他们可能感兴趣的CD。

下面这几个关键抽象在顶级包中都定义有相应的Mahout接口。

- **DataModel**：表示与用户及其项目偏好相关的信息仓库。
- **UserSimilarity**：定义两个用户之间的相似度。
- **ItemSimilarity**：定义两个项目之间的相似度。
- **UserNeighborhood**：为给定用户计算邻近用户。
- **Recommender**：为用户推荐项目。

6

图6-5显示了这些概念的总体结构。

### 1. 基于用户的过滤

通过初始化前面提到的组件，可以实现一个最基本的基于用户的协同过滤器，具体步骤如下。

首先，加载数据模型：

```
StringItemIdFileDataModel model = new StringItemIdFileDataModel(
    new File("/datasets/chap6/BX-Book-Ratings.csv", ";");
```

接着，定义计算用户关联性的方法，比如使用皮尔逊相关系数：

```
UserSimilarity similarity =
  new PearsonCorrelationSimilarity(model);
```

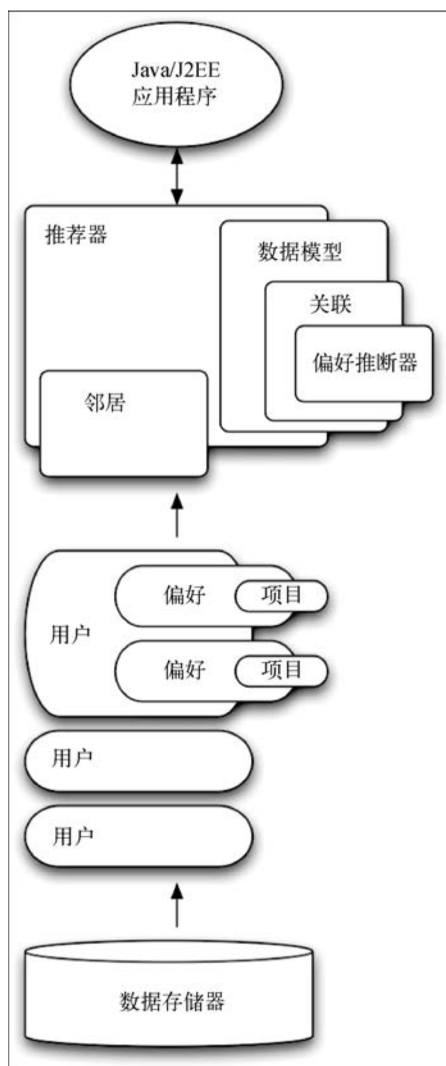



图6-5　关键抽象总体结构

然后，定义如何指出哪些用户是相似的，即评分彼此相近的用户。

```
UserNeighborhood neighborhood =
  new ThresholdUserNeighborhood(0.1, similarity, model);
```

接下来，使用数据模型、邻居、相似对象初始化GenericUserBasedRecommender默认引擎，代码如下：

```
UserBasedRecommender recommender =
new GenericUserBasedRecommender(model, neighborhood, similarity);
```

以上就是全部代码，至此，第一个最基本的推荐引擎就做好了。下面讨论应该如何调用推荐引擎。首先，打印用户已经评分的项目，以及提供给这位用户的10个推荐项目。

```
long userID = 80683;
int noItems = 10;

List<RecommendedItem> recommendations = recommender.recommend(
  userID, noItems);

System.out.println("Rated items by user:");
for(Preference preference : model.getPreferencesFromUser(userID)) {
  // convert long itemID back to ISBN
  String itemISBN = model.getItemIDAsString(
  preference.getItemID());
  System.out.println("Item: " + books.get(itemISBN) +
    " | Item id: " + itemISBN +
    " | Value: " + preference.getValue());
}

System.out.println("\nRecommended items:");
for (RecommendedItem item : recommendations) {
  String itemISBN = model.getItemIDAsString(item.getItemID());
  System.out.println("Item: " + books.get(itemISBN) +
    " | Item id: " + itemISBN +
    " | Value: " + item.getValue());
}
```



上面代码输入如下推荐项目及其分数：

```
Rated items:
Item: The Handmaid's Tale | Item id: 0395404258 | Value: 0.0
Item: Get Clark Smart : The Ultimate Guide for the Savvy Consumer | Item id: 1563526298
| Value: 9.0
Item: Plum Island | Item id: 0446605409 | Value: 0.0
Item: Blessings | Item id: 0440206529 | Value: 0.0
Item: Edgar Cayce on the Akashic Records: The Book of Life | Item id: 0876044011 | Value:
0.0
Item: Winter Moon | Item id: 0345386108 | Value: 6.0
Item: Sarah Bishop | Item id: 059032120X | Value: 0.0
Item: Case of Lucy Bending | Item id: 0425060772 | Value: 0.0
Item: A Desert of Pure Feeling (Vintage Contemporaries) | Item id: 0679752714 | Value:
0.0
Item: White Abacus | Item id: 0380796155 | Value: 5.0
Item: The Land of Laughs : A Novel | Item id: 0312873115 | Value: 0.0
Item: Nobody's Son | Item id: 0152022597 | Value: 0.0
Item: Mirror Image | Item id: 0446353957 | Value: 0.0
Item: All I Really Need to Know | Item id: 080410526X | Value: 0.0
Item: Dreamcatcher | Item id: 0743211383 | Value: 7.0
Item: Perplexing Lateral Thinking Puzzles: Scholastic Edition | Item id: 0806917695
| Value: 5.0
Item: Obsidian Butterfly | Item id: 0441007813 | Value: 0.0

Recommended items:
```

```
Item: Keeper of the Heart | Item id: 0380774933 | Value: 10.0
Item: Bleachers | Item id: 0385511612 | Value: 10.0
Item: Salem's Lot | Item id: 0451125452 | Value: 10.0
Item: The Girl Who Loved Tom Gordon | Item id: 0671042858 | Value: 10.0
Item: Mind Prey | Item id: 0425152898 | Value: 10.0
Item: It Came From The Far Side | Item id: 0836220730 | Value: 10.0
Item: Faith of the Fallen (Sword of Truth, Book 6) | Item id: 081257639X | Value: 10.0
Item: The Talisman | Item id: 0345444884 | Value: 9.86375
Item: Hamlet | Item id: 067172262X | Value: 9.708363
Item: Untamed | Item id: 0380769530 | Value: 9.708363
```

2. 基于项目的过滤

**项目相似性（ItemSimilarity）**是接下来要讨论的重点。基于项目的推荐器很有用，因为它们能够充分利用项目本身之间的关系：它们将计算建立在项目的相似性而非用户的相似性之上，项目的相似性是相对稳定的。项目的相似性可以预先计算，不需要实时重新计算。

因此，如果打算使用ItemSimilarity这个类，强烈建议使用`GenericItemSimilarity`类，这个类可以预先计算项目的相似性。也可以使用`PearsonCorrelationSimilarity`类，这个类会实时计算相似性。但对于大量数据，你会发现它的计算速度慢得让人难以忍受。

```
StringItemIdFileDataModel model = new StringItemIdFileDataModel(
  new File("datasets/chap6/BX-Book-Ratings.csv"), ";");

ItemSimilarity itemSimilarity = new
  PearsonCorrelationSimilarity(model);

ItemBasedRecommender recommender = new
  GenericItemBasedRecommender(model, itemSimilarity);

String itemISBN = "0395272238";
long itemID = model.readItemIDFromString(itemISBN);
int noItems = 10;
List<RecommendedItem> recommendations =
  recommender.mostSimilarItems(itemID, noItems);

System.out.println("Recommendations for item:
  "+books.get(itemISBN));

System.out.println("\nMost similar items:");
for (RecommendedItem item : recommendations) {
  itemISBN = model.getItemIDAsString(item.getItemID());
  System.out.println("Item: " + books.get(itemISBN) + " | Item id:
    " + itemISBN + " | Value: " + item.getValue());
}
```

```
Recommendations for item: Close to the Bone

Most similar items:
Item: Private Screening | Item id: 0345311396 | Value: 1.0
Item: Heartstone | Item id: 0553569783 | Value: 1.0
```

```
Item: Clockers / Movie Tie In | Item id: 0380720817 | Value: 1.0
Item: Rules of Prey | Item id: 0425121631 | Value: 1.0
Item: The Next President | Item id: 0553576666 | Value: 1.0
Item: Orchid Beach (Holly Barker Novels (Paperback)) | Item id: 0061013412 | Value:
1.0
Item: Winter Prey | Item id: 0425141233 | Value: 1.0
Item: Night Prey | Item id: 0425146413 | Value: 1.0
Item: Presumed Innocent | Item id: 0446359866 | Value: 1.0
Item: Dirty Work (Stone Barrington Novels (Paperback)) | Item id:
  0451210158 | Value: 1.0
```

上述输出结果中，可以看到输出的一组项目与我们选择的特定项目是相似的。

#### 3. 添加自定义规则到推荐引擎

我们经常会遇到这样的问题：一些业务规则需要提高所选项目的分数。比如，图书数据集中新到了一本书，我们想给它一个更高的分数。对此，可以使用IDRescorer接口实现完成这个任务。

- rescore(long, double)：从参数接收itemID与原始分数，返回调整后的分数。
- isFiltered(long)：若推荐不包含指定项目，则返回true，否则返回false。

对于我们的示例，可做如下实现：

```
class MyRescorer implements IDRescorer {

  public double rescore(long itemId, double originalScore) {
    double newScore = originalScore;
    if(bookIsNew(itemId)){
      originalScore *= 1.3;
    }
    return newScore;
  }

  public boolean isFiltered(long arg0) {
    return false;
  }

}
```

调用recommender.recommend时，返回一个IDRescorer实例：

```
IDRescorer rescorer = new MyRescorer();
List<RecommendedItem> recommendations =
recommender.recommend(userID, noItems, rescorer);
```

#### 4. 评估

你可能想知道，如何才能保证推荐引擎返回的推荐项目是靠谱的。准确检测推荐有效程度的唯一方法就是，在拥有实际用户的真实系统中做A/B测试。比如，A组收到一个随机推荐的项目，而B组收到我们的推荐引擎推荐的项目。

由于这并非总是可行的，也不实际，所以可以使用脱机统计评估进行估计。一种方法是使用第1章介绍的k折交叉验证。将数据集分成多个子集，其中一些用来训练我们的推荐引擎，其余的测试它对未知用户的推荐效果。

Mahout实现了RecommenderEvaluator类，这个类将一个数据集划分成两部分，第一部分默认为数据的90%，用于生成推荐；其余部分则与评估的偏好值做比较，以测试匹配效果。这个类不直接接受recommender对象，需要创建一个类实现RecommenderBuilder接口，它为一个给定的DataModel对象（稍后用于测试）创建一个recommender对象。接下来，让我们看看如何实现。

首先，创建一个实现RecommenderBuilder接口的类。需要实现buildRecommender方法，它会返回一个recommender，如下所示：

```
public class BookRecommender implements RecommenderBuilder  {
  public Recommender buildRecommender(DataModel dataModel) {
    UserSimilarity similarity =
      new PearsonCorrelationSimilarity(model);
    UserNeighborhood neighborhood =
      new ThresholdUserNeighborhood(0.1, similarity, model);
    UserBasedRecommender recommender =
      new GenericUserBasedRecommender(
        model, neighborhood, similarity);
    return recommender;
  }
}
```

有了返回recommender对象的类后，我们就可以对RecommenderEvaluator的实例做初始化。这个类的默认实现是AverageAbsoluteDifferenceRecommenderEvaluator类，用于在用户的预测评分与实际评分之间计算平均绝对差值。下面代码显示了如何将上面这些内容组合在一起，以进行Hold-Out测试。

首先，加载数据模型：

```
DataModel dataModel = new FileDataModel(
  new File("/path/to/dataset.csv"));
```

接着，初始化evaluator实例：

```
RecommenderEvaluator evaluator =
  new AverageAbsoluteDifferenceRecommenderEvaluator();
```

初始化BookRecommender对象，实现RecommenderBuilder接口：

```
RecommenderBuilder builder = new MyRecommenderBuilder();
```

最后，调用evaluate()方法，该方法接收如下参数。

- RecommenderBuilder：该对象实现了RecommenderBuilder，用于创建待测试的recommender。
- DataModelBuilder：要使用的DataModelBuilder，若为null，将使用默认的DataModel实现。
- DataModel：用于测试的数据集。
- trainingPercentage：表示生成推荐的每个用户偏好所占的比例，其他的则与估计的偏好值做比较，以评估recommender的性能。
- evaluationPercentage：评估中参与的用户在数据模型中所占的比例。

evaluate()方法调用如下：

```
double result = evaluator.evaluate(builder, null, model, 0.9,
  1.0);
System.out.println(result);
```

evaluate()方法返回一个double值，0表示评估结果最佳，表明推荐器完美匹配用户偏好。一般来说，值越小，匹配得越好。

#### 5. 在线学习引擎

在线学习引擎中的“在线”两字是什么意思呢？上面讲到的推荐引擎对于已有的用户有很好的工作效果，但对于那些新注册的用户推荐效果不佳。我们肯定也想为这些新用户做一些合理的推荐。创建一个推荐实例代价很大（它肯定比一个普通的网络请求需要花更长时间），因此不能每次都创建一个新推荐。

6

幸运的是，Mahout允许我们向数据模型添加临时用户。一般设置如下：

- 使用当前数据定期重建整个推荐，比如每天或每小时，具体取决于耗费多长时间。
- 做推荐时，检查系统中是否有这个用户。
- 若有，像往常一样结束推荐。
- 若没有，则创建临时用户，填入偏好，并做推荐。

如果你的内存有限，第一部分（定期重建推荐器）其实很难办到：创建新推荐器时，你需要在内存中保留数据的两个副本（为了可以处理来自旧推荐器的请求）。然而，由于这对推荐真的没什么用，所以就不细讲了。

对于临时用户，可以使用一个PlusAnonymousConcurrentUserDataModel类实例包装我们的数据模型。这个类允许获得一个临时用户ID，以后必须释放这个ID，以便可以重用（这样的ID数目是有限制的）。得到ID后，必须填写偏好，然后可以像以前一样开始推荐：

```
class OnlineRecommendation{

  Recommender recommender;
```

```
  int concurrentUsers = 100;
  int noItems = 10;

  public OnlineRecommendation() throws IOException {


    DataModel model = new StringItemIdFileDataModel(
      new File /chap6/BX-Book-Ratings.csv"), ";");
    PlusAnonymousConcurrentUserDataModel plusModel = new
      PlusAnonymousConcurrentUserDataModel
        (model, concurrentUsers);
    recommender = ...;

  }

  public List<RecommendedItem> recommend(long userId,
    PreferenceArray preferences){

    if(userExistsInDataModel(userId)){
      return recommender.recommend(userId, noItems);
    }

    else{

      PlusAnonymousConcurrentUserDataModel plusModel =
        (PlusAnonymousConcurrentUserDataModel)
          recommender.getDataModel();

      // 从poll获取一个可用的匿名用户
      Long anonymousUserID = plusModel.takeAvailableUser();

      // 设置临时偏好
      PreferenceArray tempPrefs = preferences;
      tempPrefs.setUserID(0, anonymousUserID);
      tempPrefs.setItemID(0, itemID);
       plusModel.setTempPrefs(tempPrefs, anonymousUserID);

      List<RecommendedItem> results =
        recommender.recommend(anonymousUserID, noItems);

      // 释放用户到poll
      plusModel.releaseUser(anonymousUserID);

      return results;

    }

  }
}
```

## 6.4　基于内容的过滤

Mahout框架不包含基于内容的过滤，主要是因为，如何定义相似项目是由你自己决定的。如果想定义一个基于内容的项目到项目相似度，需要实现自己的ItemSimilarity。比如，我们的图书数据集中，针对图书相似度，可能制定如下规则：

- 若类型相同，则将相似度加0.15；
- 若作者相同，则将相似度加0.50。

下面实现我们自己的相似度测度，如下：

```
class MyItemSimilarity implements ItemSimilarity {
 ...
 public double itemSimilarity(long itemID1, long itemID2) {
  MyBook book1 = lookupMyBook (itemID1);
  MyBook book2 = lookupMyBook (itemID2);
  double similarity = 0.0;
  if (book1.getGenre().equals(book2.getGenre())
   similarity += 0.15;
  }
  if (book1.getAuthor().equals(book2. getAuthor ())) {
   similarity += 0.50;
  }
  return similarity;
 }
 ...
}
```

然后，使用这个ItemSimilarity替换LogLikelihoodSimilarity或其他GenericItem-Based Recommender的实现。以上就是在Mahout框架中做基于内容的推荐。

此处给出的示例是基于内容推荐的一种最简单的形式。还有一种方法可以用来创建基于内容的用户画像（content-based profile of users），它建立在项目特征的加权向量基础之上。权重表示每个特征对于用户的重要程度，这可以从单独评分的内容向量计算出来。

## 6.5　小结

本章学习了推荐引擎的基本概念、协同过滤与基于内容的过滤之间的不同，以及如何使用Apache Mahout。它是创建推荐器的基础框架，拥有良好的可配置性，并且提供大量扩展点。此外还学习了如何选择正确的配置参数值、建立分值重计算，并对推荐结果进行评估。

本章应用数据科学技术分析顾客行为，先用了第4章中讲解的客户关系预测，然后使用了第5章中的关联分析。下一章将继续学习另一主题——欺诈与异常检测。

# 第 7 章 欺诈与异常检测

异常检测（outlier detection）用于识别异常、罕见事件或其他反常情况。查找这样的异常就像大海捞针，但它们可能招致令人相当震惊的后果，比如信用卡欺诈检测、网络入侵识别、制造工艺缺陷、临床试验、投票活动、电子商务犯罪。因此，及时发现这些异常可能避免巨大损失。应用机器学习检测异常问题会带来新的发现，并且能够得到更好的异常事件检测效果。机器学习能够考虑许多不同的数据来源，并找到人类分析时难以发现的关系。

以电子商务欺诈检测为例，通过适当应用机器学习算法，进行欺诈检测时，我们可以把购物者的在线行为（即网站浏览历史）列入欺诈检测算法的考察对象，这样判断时考虑的因素更全面，而不只是简单地考虑持卡者的购物历史。这包括分析各种数据源，对电子商务欺诈检测而言，它还是一种更加稳健的方法。

本章涵盖如下主题：

- 问题与挑战
- 可疑模式检测
- 异常模式检测
- 使用不平衡数据集
- 时序中的异常检测

## 7.1 可疑与异常行为检测

从传感器数据学习模式的问题会出现在许多应用场合，包括电子商务、智能环境、视频监控、网络分析、人机交互、环境辅助智能系统等。我们的重点是检测那些有悖于常规行为的模式，它们可能是安全风险、健康问题或者其他任何异常行为偶发事件。

换言之，异常行为是一种数据模式，它要么与期望的行为不符（异常行为），要么是一个先前定义的不希望发生的行为（可疑行为）。异常行为模式包括异常值（outlier）、异常（exception）、特性（peculiarity）、意外（surprise）、滥用（misuse）等。相对而言，这些模式并不会经常出现，但确实在某些时候会出现，有可能产生令人吃惊的后果，并且这些后果一般都是负面的。典型的

例子有信用卡欺诈检测、网络入侵、工业破坏。电子商务中，欺诈给商人造成的损失每年超过200亿美元；在医疗保健领域，欺诈每年耗掉纳税人60亿美元的税费；在银行领域，欺诈造成的损失超过12亿美元。

### 未知的未知

2002年2月12日，时任美国国防部长唐纳德·拉姆斯菲尔德在召开的新闻发布会上称，没有证据表明伊拉克政府向恐怖分子提供了大规模杀伤性武器。这立刻引发了人们的热议。拉姆斯菲尔德声称（DoD News，2012）：

> “那些对还未发生的事所做的报道总能引起我们的兴趣，正如我们知道，有些事情大家都知道（known knowns），即有些事情我们知道我们知道。我们也都知道有些事情我们不知道（known unknowns），那就是说，我们知道有些事情我们不知道。但也有些事情我们都不知道（unknown unknowns），就是那些我们不知道我们不知道的事情。纵观我们国家与其他自由国家的历史可以发现，后面那些往往是最难的。”

乍看上去，这段声明可能有点绕，但是“未知的未知”（unknown unknowns）这个想法在处理风险的专家、NSA和其他情报机构中得到深入研究。上面这段声明基本意思如下。

- **已知的已知（Known-knowns）**：这些问题已为我们所熟知，我们知道如何辨认它们，也知道如何处理它们。
- **已知的未知（Known-unknowns）**：这些问题可以预料或预见到，我们可以做出合理预测，但是它们之前从未发生过。
- **未知的未知（Unknown-unknowns）**：这些问题是不可预料的，也无法预见，我们不能根据以往经验做预测，它们将我们置于重大风险之中。



接下来，我们将学习两种处理前两种已知与未知的基本方法：可疑模式检测（处理已知的已知）与异常模式检测（处理已知的未知）。

## 7.2 可疑模式检测

第一种方法假设有一个行为库，它对图7-1用减号表示的负模式（negative pattern）进行编码。这样，识别被观察行为就转换成从库中找出一个对应的匹配。如果发现一个新模式（圆圈）与负模式不匹配，那么就把这个新模式看作是可疑的。

比如，你生病看医生时，针对病情，她会检查各种症状（体温、疼痛程度、影响范围等），然后将这些症状与已知的疾病进行匹配。使用机器学习的术语来说就是，医生收集属性，并做分类处理。

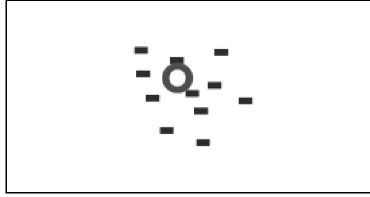


图7-1　可疑模式检测

这个方法的优点在于，我们可以立刻知道问题是什么。假如我们了解那种疾病，就能给出合适的治疗方法。

这个方法的主要缺点是，它只能检测到那些我们事先已经知道的可疑模式。如果一个模式在负模式库（negative pattern library）中不存在，那么将无法识别。因此，这个方法适合为那些“已知的已知”建模。

## 7.3　异常模式检测

第二个方法是以相反的方式使用模式库，即模式库只对正模式（图7-2中的加号）做编码。当一个被观察的行为（圆圈）与模式库中的所有模式都不匹配时，就会被视作异常。

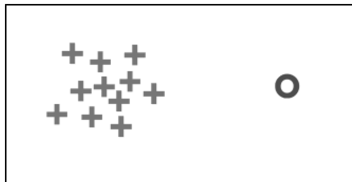


图7-2　异常模式检测

这个方法只需要我们为过去所见建模，即为那些正常模式（normal patterns）建模。回到看病的例子，我们之所以去看医生主要是因为感觉自己身体不舒服。生病时的感觉（比如头痛、疼痛）与平时的感觉不一样，所以我们才决定去看医生。我们不知道什么疾病引起了这些症状，也不知道该如何治疗，但我们能明显感觉到这与平时的感觉完全不一样。

这个方法的主要优点是，它不需要我们提及非正常模式（non-normal patterns），因此很适合用来为“已知的未知”与“未知的未知”建模。另一方面，这个方法不能准确指出问题是什么。

### 7.3.1　分析类型

不管怎样，我们有多种方法可以进行类型分析。如下3种类型（模式分析、事务分析、规划识别）中，我们会对异常与可疑行为检测做大致分类。接下来，我们将快速了解一些实际生活中

的应用程序。

**模式分析**

基于视觉方式（比如摄像机），根据行为模式做异常与可疑行为检测是一个在计算视觉研究中相对活跃的领域。2007年，Zhang等人提出了一个系统，根据视频序列做人体运动视觉分析，它可以根据步态轨迹识别异常行为；2009年，Lin等人描述了一个基于颜色特征、距离特征、计数特征的视频监控系统，使用演化技术度量观察的相似性。该系统追踪每个人，分析他们的轨迹模式，进而对其行为进行分类。这个系统会从图像的不同部分抽取一组低层视觉特征，使用SVM算法进行分类，以检测用户的攻击、快乐、醉酒、焦虑、中立、疲倦行为。

### 7.3.2 事务分析

不同于连续观察，事务分析关注的是离散状态与事务，主要研究领域是**入侵检测**（**ID**），一般目标是检测那些攻击信息系统的入侵行为。ID系统主要分为两种类型，一种是基于特征的（signature-based），另一种是基于异常的（anomaly-based）。如前所述，它主要应用于可疑与异常模式检测。如果想进一步了解有关ID的内容，建议阅读Gyanchandani等人2012年合写的图书。

而且，环境辅助智能系统中，基于可穿戴传感器的应用也适应于事务分析，因为感知通常是基于事件的。2008年，Lymberopoulos等人提出了一个从用户家中设置的传感器网络自动提取用户时序斑图（spatio-temporal patterns）的系统，这些时序斑图被编码成传感器触发（sensor activations）。他们提出的方法基于位置、时间、持续时间，能够使用Apriori算法提取频繁模式，并且将最频繁的模式编码为马尔可夫链。另一个相关领域是**隐马尔可夫模型**（**Hidden Markov Models，HMM**，Rabiner，1989），它对行为序列建模，广泛应用于习惯行为识别。这些内容已经超出本书讨论的范围，不再详述。

### 7.3.3 规划识别

规划识别主要关注于识别智能体不可观测状态的机制，前提是给出其与环境交互的观察结果（AvrahamiZilberbrand，2009）。大部分现存的调查研究都认为，行为的观察结果是离散的。为了进行异常与可疑行为检测，规划识别算法可能使用混合方法：符号规划识别器可以用于筛选一致的假设，将其传递给一个注重排名的评价引擎。

这些先进方法被应用于各种真实生活场景，以发现异常。接下来，我们将学习更多用于检测可疑与异常模式的方法。

## 7.4 保险理赔欺诈检测

首先了解可疑行为检测，它的目标是学习未知欺诈模式，这些模式符合“已知的已知”模型。

### 7.4.1　数据集

我们将使用一个描述保险交易的数据集，可以在**Oracle Database Online Documentation**（2015）中找到它，网址如下：

http://docs.oracle.com/cd/B28359_01/datamine.111/b28129/anomalies.htm

这个数据集描述了一个未公开的保险公司的车辆事故保险理赔情况。其中包含15 430个理赔案例，每个理赔用33个属性进行描述：

- 客户详细信息（**年龄**、**性别**、**婚姻状况**等）
- 购买保单（**保单种类**、**车辆类型**、增补数、代理类型等）
- 理赔情况（日/月/周理赔，保单报告归档、证人、事故报告间隔日期、事故理赔等）
- 其他客户数据（车辆数、上一次理赔、**司机评级**等）
- 有无欺诈（有或无）

图7-3显示的是其中一个样本，是把数据加载到Weka之后显示的画面。

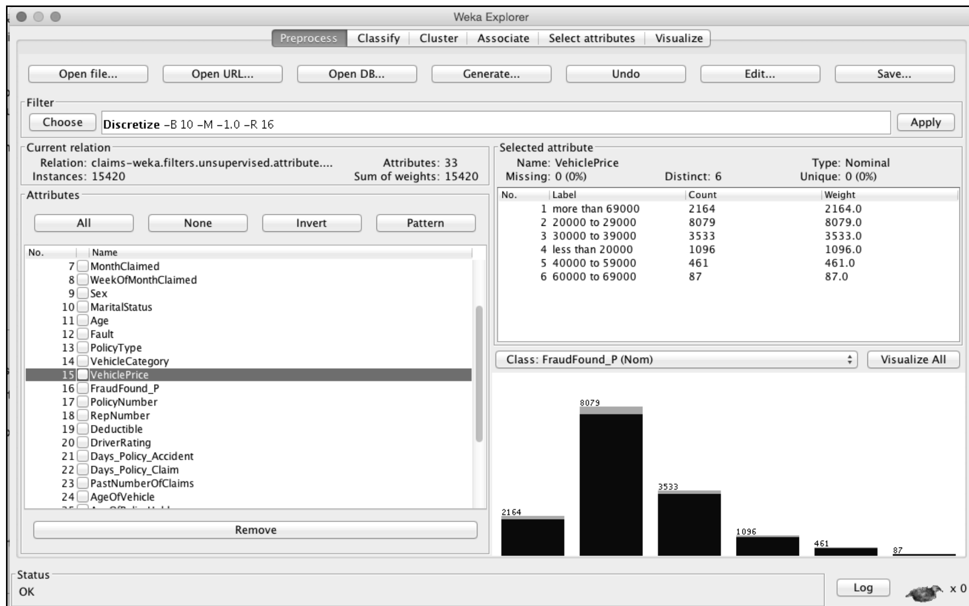



图7-3　将数据加载到Weka

我们的任务是创建一个模型，将来用于识别可疑的理赔。这项任务面临的挑战是，这些索赔中只有6%是可疑的。如果我们创建了一个虚拟分类器，说所有索赔都不可疑，那在94%的索赔案例中它的判断都是对的。因此，这项任务中，我们将使用不同的精确度测量方法：准确率与召回率。

回想表1-3，表格中有4种可能的结果，分别是真正、假正、假负、真负，由此可推出表7-1。

表7-1　可能结果的可能分类

| | | 分类为 | |
|---|---|---|---|
| 真实情况 | | 欺诈 | 非欺诈 |
| | 欺诈 | TP-真正 | FN-假负 |
| | 非欺诈 | FP-假正 | TN-真负 |

准确率与召回率定义如下。

- 准确率等于在被分类器判定为正的所有样本实例（TP+FP）中，正确判断为正（TP）的正例样本所占比重，如下：

$$\mathrm{Pr} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}$$

- 召回率等于在总正例样本（TP+FN）中，被正确判定为正（TP）的正例所占比重，如下：

$$\mathrm{Re} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

- 若使用这些指标，我们的虚拟分类器得分是：Pr=0、Re=0，因为它从不会将任何实例标记为欺诈（TP=0）。实际上，我们希望使用这两个指标评价分类器，即*F*值（F-measure），它是一种在准确率与召回率之间计算调和平均数的方法，如下：

$$F\text{值} = \frac{2 * \mathrm{Pr} * \mathrm{Re}}{\mathrm{Pr} + \mathrm{Re}}$$

接下来，继续设计一个真实的分类器。

### 7.4.2　为可疑模式建模

为了设计一个分类器，我们将遵循第1章讲解的有关监督学习的标准步骤。此处还要加入其他一些步骤，以处理非平衡数据集，并对基于准确率与召回率的分类模型做评估。详细步骤如下。

- 加载.csv格式的数据；
- 指派分类属性；
- 将所有属性从数值型（numeric）转换为名义型（nominal），确保没有错误加载的数值；
- **Experiment 1**：使用*k*折交叉验证评价模型；
- **Experiment 2**：重新调整数据集，使之拥有更均衡的类分布，并且手动交叉验证；
- 使用召回率、精确率与*F*值评价分类器。

首先，使用CSVLoader类加载数据，代码如下：

```
String filePath = "/Users/bostjan/Dropbox/ML Java
Book/book/datasets/chap07/claims.csv";

CSVLoader loader = new CSVLoader();
loader.setFieldSeparator(",");
loader.setSource(new File(filePath));
Instances data = loader.getDataSet();
```

然后，确保所有属性的类型都是名义型。在数据导入期间，Weka会使用一些启发式方法猜测属性最有可能的类型，即数值型、名义型、字符串型、日期型。由于启发式方法并非总能正确猜中属性类型，所以要手动设置属性类型，代码如下：

```
NumericToNominal toNominal = new NumericToNominal();
toNominal.setInputFormat(data);
data = Filter.useFilter(data, toNominal);
```

继续下一步之前，还要指定待预测的属性。可以通过调用setClassIndex(int)函数实现。

```
int CLASS_INDEX = 15;
data.setClassIndex(CLASS_INDEX);
```

接着，剔除一个描述保单编号的属性，因为它不具有预测价值。只要简单使用Remove过滤器即可移除。

```
Remove remove = new Remove();
remove.setInputFormat(data);
remove.setOptions(new String[]{"-R", ""+POLICY_INDEX});
data = Filter.useFilter(data, remove);
```

以上就是建模的准备工作。

### 1. 纯方法

纯方法（Vanilla approach）是指直接应用第3章介绍的方法，既不需要做预处理，也不需要考虑数据集具体细节。为了说明Vanilla方法的缺点，我们将创建一个带有默认参数的简单模型，并且应用k折交叉验证。

首先，添加要测试的分类器：

```
ArrayList<Classifier>models = new ArrayList<Classifier>();
models.add(new J48());
models.add(new RandomForest());
models.add(new NaiveBayes());
models.add(new AdaBoostM1());
models.add(new Logistic());
```

然后，创建一个Evaluation对象，通过调用crossValidate(Classifier, Instances, int, Random, String[])方法进行$k$折交叉验证，最后输出准确率、召回率与$F$值。

```
int FOLDS = 3;
Evaluation eval = new Evaluation(data);

for(Classifier model : models){
  eval.crossValidateModel(model, data, FOLDS,
  new Random(1), new String[] {});
  System.out.println(model.getClass().getName() + "\n"+
    "\tRecall:    "+eval.recall(FRAUD) + "\n"+
    "\tPrecision: "+eval.precision(FRAUD) + "\n"+
    "\tF-measure: "+eval.fMeasure(FRAUD));
}
```

最终输出的评估分数如下：

```
weka.classifiers.trees.J48
  Recall:    0.03358613217768147
  Precision: 0.9117647058823529
  F-measure: 0.06478578892371996
...
weka.classifiers.functions.Logistic
  Recall:    0.037486457204767065
  Precision: 0.2521865889212828
  F-measure: 0.06527070364082249
```

从上面这些输出可以看到，结果并不理想。从召回率（Recall）看，所有欺诈行为中，被发现的欺诈只占1%~3%。也就是说，只有1~3/100个欺诈能被检测到。而从准确率（Precision）来说，报警准确率有91%。这意味着申请理赔的9/10个案例中，模型都是正确的。

2. 数据集重整

由于相比于正例，反例（即欺诈）数目非常少，所以学习算法要与归纳推理做“斗争”。为帮助算法解决这个问题，我们可以给它们一个数据集。该数据集中，正例与反例的比重是可比的。也就是说，这个问题可以通过数据集重整（Dataset rebalancing）得到解决。



Weka有一个内置的过滤器**Resample**，用于从一个数据集随机抽选子样本，它使用重置抽样或不重置抽样。这个过滤器也可以将一个分布调整为类均匀分布。

我们将手工实现*k*折交叉验证。首先，把数据集均等地分成*k*折，其中第*k*折用作测试，其他折用来学习。使用`StratifiedRemoveFolds`过滤器划分数据集，划分后的各折中仍然保持着相同的类分布，如下：

```
StratifiedRemoveFolds kFold = new StratifiedRemoveFolds();
kFold.setInputFormat(data);

double measures[][] = new double[models.size()][3];

for(int k = 1; k <= FOLDS; k++){

  // 把数据划分为测试折与训练折
```

```
  kFold.setOptions(new String[]{
    "-N", ""+FOLDS, "-F", ""+k, "-S", "1"});
  Instances test = Filter.useFilter(data, kFold);

  kFold.setOptions(new String[]{
    "-N", ""+FOLDS, "-F", ""+k, "-S", "1", "-V"});
    // 反选-V
  Instances train = Filter.useFilter(data, kFold);
```

然后，重整训练数据集，-Z参数指定要重抽样数据集的比例，-B把类分布调整成均匀分布：

```
Resample resample = new Resample();
resample.setInputFormat(data);
resample.setOptions(new String[]{"-Z", "100", "-B", "1"}); //with
  replacement
Instances balancedTrain = Filter.useFilter(train, resample);
```

接着，创建分类器并做评估。

```
for(ListIterator<Classifier>it = models.listIterator();
  it.hasNext();){
  Classifier model = it.next();
  model.buildClassifier(balancedTrain);
  eval = new Evaluation(balancedTrain);
  eval.evaluateModel(model, test);

// 为计算平均数保存结果
  measures[it.previousIndex()][0] += eval.recall(FRAUD);
  measures[it.previousIndex()][1] += eval.precision(FRAUD);
 measures[it.previousIndex()][2] += eval.fMeasure(FRAUD);
}
```

最后，计算平均数，输出最佳模型：

```
// 计算平均数
for(int i = 0; i < models.size(); i++){
  measures[i][0] /= 1.0 * FOLDS;
  measures[i][1] /= 1.0 * FOLDS;
  measures[i][2] /= 1.0 * FOLDS;
}

// 输出结果，选择最佳模型
Classifier bestModel = null; double bestScore = -1;
for(ListIterator<Classifier> it = models.listIterator();
  it.hasNext();){
  Classifier model = it.next();
  double fMeasure = measures[it.previousIndex()][2];
  System.out.println(
    model.getClass().getName() + "\n"+
    "\tRecall:    "+measures[it.previousIndex()][0] + "\n"+
    "\tPrecision: "+measures[it.previousIndex()][1] + "\n"+
    "\tF-measure: "+fMeasure);
  if(fMeasure > bestScore){
    bestScore = fMeasure;
```

```
      bestModel = model;

    }
  }
  System.out.println("Best model:"+bestModel.getClass().getName());
```

现在，模型性能得到显著提升，如下：

```
weka.classifiers.trees.J48
  Recall:    0.44204845100610574
  Precision: 0.14570766048577555
  F-measure: 0.21912423640160392
...
weka.classifiers.functions.Logistic
  Recall:    0.7670657247204478
  Precision: 0.13507459756495374
  F-measure: 0.22969038530557626
Best model: weka.classifiers.functions.Logistic
```

从上述结果可以看到，所有模型的得分都有了显著提高，比如最佳模型——逻辑回归，它发现欺诈的准确率达到了76%；而误报率相对合理，被标记为"欺诈"的索赔中，只有13%是真正的欺诈。如果"漏掉一个坏人"（未检测到欺诈行为）所付出的代价要比"误杀一个好人"（欺诈误报）的代价大得多，那么"宁可误杀也不错放"就是理所应当的了。

模型的整体性能可能还有一些提升空间，比如我们可以做属性选择与特征生成，并且应用更复杂的模型学习，相关内容已经在第3章讲解过。

## 7.5　网站流量异常检测



第二个例子中，我们将对前一个例子的反面建模。我们不会讨论典型的少欺诈案例是什么，而要讨论系统正常的预期行为。如果有违背模型预期的事情发生，这个事情就会被识别为异常。

### 7.5.1　数据集

我们将使用一个Yahoo Labs公开的数据集，这有助于我们研究如何从时序数据中检测异常。对于Yahoo而言，主要使用案例是检测Yahoo服务器上的非正常流量。

尽管Yahoo宣称他们的数据是公开的，但使用时必须申请，一般24小时后才能获得授权。你可以从如下网址获取数据集：

http://webscope.sandbox.yahoo.com/catalog.php?datatype=s&did=70

数据集由流向Yahoo服务的真实流量组成，里面同时含有一些合成数据。这个数据集总共包含367个时间序列，每个时序包含741~1680个观察结果，它们是按固定时间间隔记录得到的。每个序列都有自己的记录文件，每一行对应一个观察结果。每个序列都带有一个列指示器——"异

常”（anomaly），观察表明这个序列是异常时，该列值为1，否则为0。实际数据中的异常由人为判断决定，而合成数据中的异常由算法自动生成。表7-2展示的是几个合成时序数据的例子。

表7-2　合成时序数据片段

| timestamp | value | anomaly | changepoint | trend | noise | 12 hour seasonality | daily seasonality | weekly seasonality |
|---|---|---|---|---|---|---|---|---|
| 1422237600 | 4333.43 | 0 | 0 | 4599 | 1.81 | –190.95 | –128.86 | 52.44 |
| 1422241200 | 4316.14 | 0 | 0 | 4602 | –14.65 | –220.5 | –105.21 | 54.51 |
| 1422244800 | 4403.20 | 0 | 0 | 4605 | 7.04 | –190.95 | –74.39 | 56.51 |
| 1422248400 | 4531.20 | 0 | 0 | 4608 | 13.52 | –110.25 | –38.51 | 58.43 |
| 1422252000 | 4967.50 | 1 | 0 | 4911 | –3.77 | –6.91 | –2.33 | 60.27 |

接下来，我们将学习如何将时序数据转换为属性，以便应用机器学习算法。

## 7.5.2　时序数据中的异常检测

在原始流式时序数据中检测异常时，需要对数据做一些转换。最明显的做法是选择一个时间窗口，用固定长度采集时间序列。接下来，我们比较新时间序列与之前采集的序列，以检测是否有异常发生。

可以选用的比较技术多种多样：

- 预测最有可能的跟随值（following value）以及置信区间（比如霍尔特-温特指数平滑）。若新值超出预测的置信区间，即被判定为异常。
- 互相关（Cross correlation）技术比较新样本与正例样本库，查找准确匹配。若未发现匹配，则把新样本标记为异常。
- 动态时间规整与互相关类似，但它允许比较中有信号失真。
- 信号离散化到频带，每个频带对应于一个字母，比如A=[min, mean/3]、B=[mean/3, mean*2/3]、C=[mean*2/3, max]，将信号转换为字母序列，比如aAABAACAABBA….这个方法可以有效减少存储，并且允许我们使用文本挖掘算法。文本挖掘相关内容将在第10章讲解。
- 基于分布的方法评估一个特定时间窗口中值的分布。观察一个新样本时，可以将其分布与之前观察的进行比较，查看是否匹配。

上述列表并不全面，这些不同方法都将重点放在检测某些异常上（比如值异常、频率异常、分布异常）。接下来将重点讲解基于分布的方法。

### 1. 基于直方图的异常检测

基于直方图的异常检测中，通过某个选定的时间窗口对信号做切分，如图7-4所示。

针对每个窗口计算直方图，也就是说，针对选中的桶数，计算每个桶中落入多少个值。直方图反映了所选时间窗口中值的分布情况（下图中间部分）。

然后，可以直接把直方图表示成实例，每个bin对应一个属性。而且，通过应用维度缩减技术（比如**主成分分析**，**PCA**），可以减少属性数目，这样就可以使用散点图绘制降维后的直方图，其中每个点代表一个直方图（下图右下）。

我们的示例中，主要的想法是先对网站流量观察几天，然后创建直方图，比如以4小时为窗口创建一个正常行为库。若新时间窗口直方图与正常行为库不匹配，就将其标记为异常。

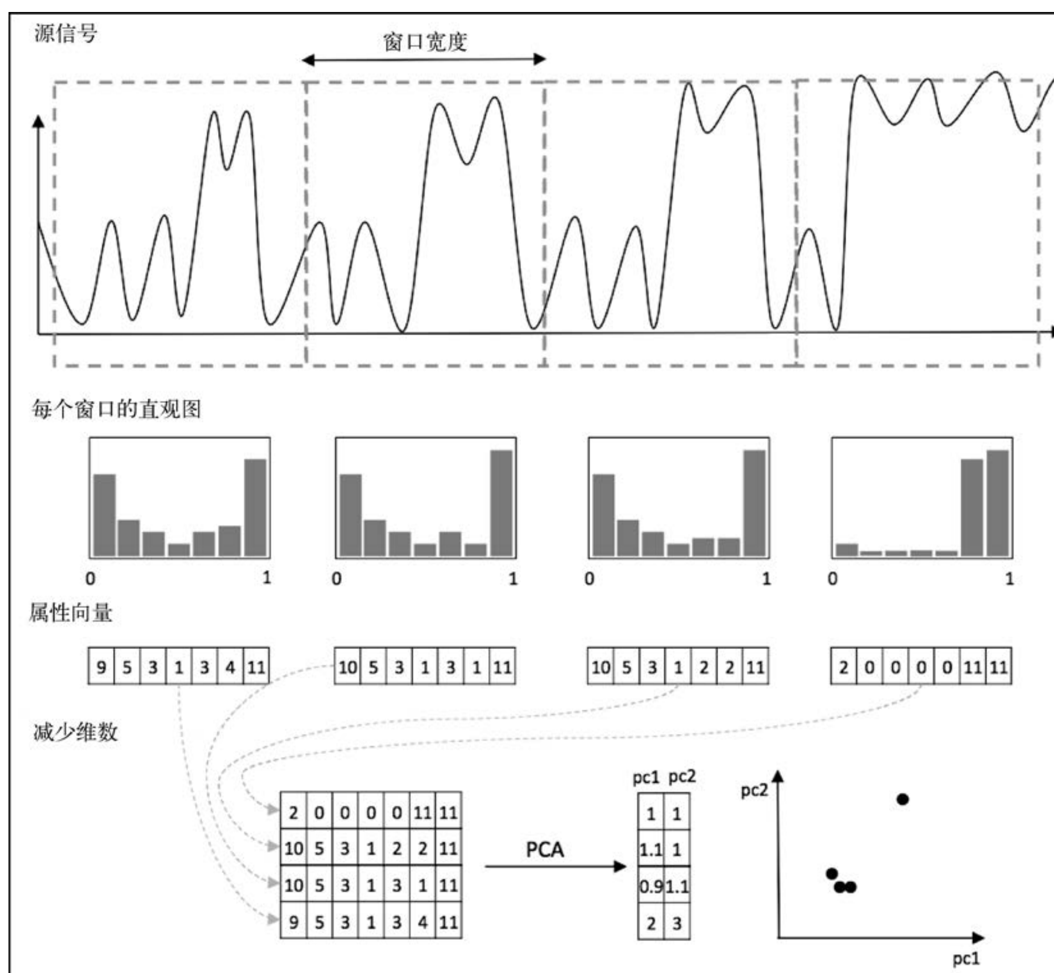



图7-4　基于直方图的异常检测

比较一个新的直方图与一组已有的直方图时，我们会使用基于密度的k最近邻算法——**局部**

7

**异常因子算法**（**LOF**，Breunig等，2000）。这个算法能够处理拥有不同密度的群组（clusters），如图7-5所示。比如，相比于左下方小而密集的群组，右上方群组既大又广。

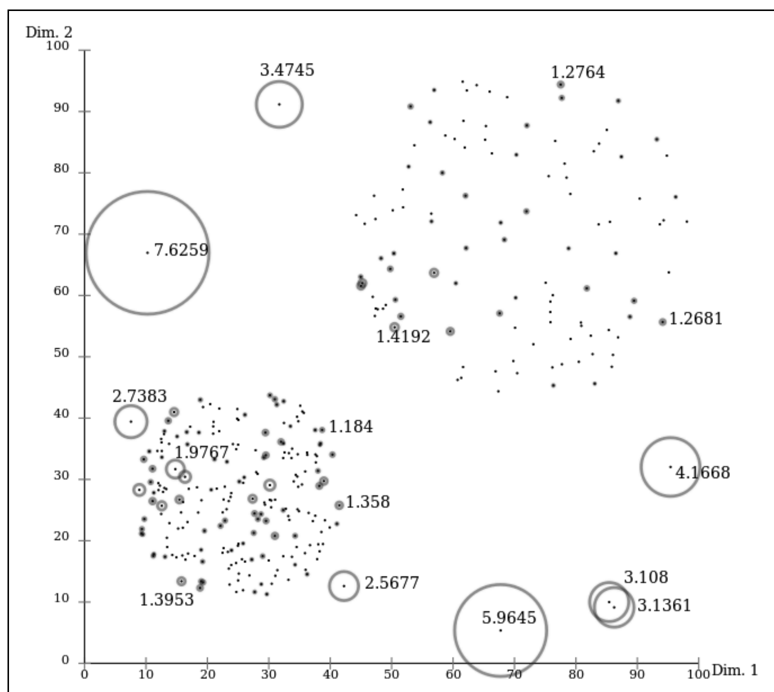



图7-5　局部异常因子算法

下面让我们开始吧！

### 2. 加载数据

第一步中，我们需要把数据从文本文件加载到一个Java对象。这些文本文件存储在一个文件夹，每个文件中，每一行就是一个带有值的时间序列。我们将其加载到一个`Double`型列表：

```
String filePath = "chap07/ydata/A1Benchmark/real";
List<List<Double>> rawData = new ArrayList<List<Double>>();
```

对直方图做正态化处理的过程中，要用到`min`与`max`值。因此数据传递时，需要先得到它们。

```
double max = Double.MIN_VALUE;
double min = Double.MAX_VALUE;

for(int i = 1; i<= 67; i++){
  List<Double> sample = new ArrayList<Double>();
  BufferedReader reader = new BufferedReader(new
    FileReader(filePath+i+".csv"));
```

```
    boolean isAnomaly = false;
    reader.readLine();
    while(reader.ready()){
      String line[] = reader.readLine().split(",");
      double value = Double.parseDouble(line[1]);
      sample.add(value);

      max = Math.max(max, value);
      min = Double.min(min, value);

      if(line[2] == "1")
        isAnomaly = true;

    }
    System.out.println(isAnomaly);
    reader.close();

    rawData.add(sample);
  }
```

至此，数据加载完毕，接下来开始创建直方图。

### 3. 创建直方图

我们将按照`WIN_SIZE`宽度为选定的时间窗口创建直方图。这个直方图用于存储`HIST_BINS`值桶（value buckets）。这些包含double列表的直方图会被存储到一个数组列表。

```
int WIN_SIZE = 500;
int HIST_BINS = 20;
int current = 0;

List<double[]> dataHist = new ArrayList<double[]>();
for(List<Double> sample : rawData){
  double[] histogram = new double[HIST_BINS];
  for(double value : sample){
    int bin = toBin(normalize(value, min, max), HIST_BINS);
    histogram[bin]++;
    current++;
    if(current == WIN_SIZE){
      current = 0;
      dataHist.add(histogram);
      histogram = new double[HIST_BINS];
    }
  }
  dataHist.add(histogram);
}
```

直方图已经做好，接下来要把它们转换为Weka中的`Instance`对象。每个直方图值对应于一个Weka属性，代码如下：

```
ArrayList<Attribute> attributes = new ArrayList<Attribute>();
for(int i = 0; i<HIST_BINS; i++){
```

```
  attributes.add(new Attribute("Hist-"+i));
}
Instances dataset = new Instances("My dataset", attributes,
  dataHist.size());
for(double[] histogram: dataHist){
  dataset.add(new Instance(1.0, histogram));
}
```

至此，我们已经加载好数据集，接下来应用异常检测算法。

#### 4. 基于密度的k最近邻算法

为了演示LOF算法（即局部异常因子算法）如何计算分数，先使用`testCV(int, int)`函数把数据集划分为训练集与测试集。其中，第一个参数用于指定折数，第二个参数指定要返回哪个折。

```
// 把数据集划分为训练集与测试集
Instances trainData = dataset.testCV(2, 0);
Instances testData = dataset.testCV(2, 1);
```

虽然LOF算法不是Weka发布版本的一部分，但我们仍然可以使用Weka的包管理器下载它。

http://weka.sourceforge.net/packageMetaData/localOutlierFactor/index.html

LOF算法有两个实现接口：一个用作无监督过滤器，计算LOF值（已知的未知）；另一个用作监督k-nn分类器（已知的已知）。我们的示例要计算异常分数因子（outlier-ness factor），所以将使用无监督过滤器接口：

```
import weka.filters.unsupervised.attribute.LOF;
```

使用常规过滤器的初始化方式对这个过滤器进行初始化。可以指定邻居的*k*数（比如`k=3`）以及`-min`与`-max`参数。LOF允许我们指定两个不同的*k*参数，在内部一个用作上界，另一个用作下界，以便查找最小/最大数`lof`值：

```
LOF lof = new LOF();
lof.setInputFormat(trainData);
lof.setOptions(new String[]{"-min", "3", "-max", "3"});
```

接下来，将训练实例加载到过滤器，用作正例库。加载完成后，调用`batchFinished()`方法对内部计算做初始化：

```
for(Instance inst : trainData){
  lof.input(inst);
}
lof.batchFinished();
```

最后，将过滤器应用于测试数据。过滤器将处理实例，并在最后添加一个包含LOF评分的属性。我们可以在控制台简单输出分数。

```
Instances testDataLofScore = Filter.useFilter(testData, lof);

for(Instance inst : testDataLofScore){
  System.out.println(inst.value(inst.numAttributes()-1));
}
```

前面几个测试实例的LOF分数如下：

```
1.306740014927325
1.318239332210458
1.0294812291949587
1.1715039094530768
```

为了理解LOF值，需要先了解LOF算法。LOF算法比较一个实例的密度与其最近邻的密度，两个分数相除就是LOF分数。若LOF分数接近1，则表示密度近似相等。LOF值越大，表示实例密度越低于它邻居的密度。这些情况下，实例会被标记为异常。

## 7.6　小结

本章学习了检测异常与可疑模式的内容。我们先讨论了两种基本方法，重点是对正模式或负模式进行编码的库。然后搞到了两个实际数据集，讨论了如何处理非平衡类分布，以及如何在时序数据中做异常检测。

下一章将深入学习各种模式，掌握创建基于模式的分类器的更高级方法，并讨论如何使用深度学习自动为图像指派标签。

第 8 章

# 利用Deeplearning4j进行图像识别

网络世界中，图像无处不在，遍及Web服务、社交网络、网店等各个领域。与人类相比，在理解图像内容以及图像含义方面，计算机遇到了很大困难。本章先介绍计算机理解图像教育方面遇到的难题，接着重点讲解一个基于深度学习的解决方法。我们会学习配置深度学习模型的高层次理论，并且讨论如何使用一个Java库——Deeplearning4j实现对图像进行分类的模型。

本章涵盖如下内容：

- 图像识别简介
- 讨论深度学习基础
- 创建一个图像识别模型

## 8.1 图像识别简介

图像识别的典型目标是从一幅数字图像中检测并识别一个对象。图像识别可以应用于工厂自动化系统，以监督产品质量；也可以应用于监控系统，以识别潜在的危险行为，比如行人或移动的车辆；还可以应用到=于安保系统，以通过指纹、虹膜、面部特征进行生物特征识别；以及应用于汽车自动驾驶技术，以重建路面与环境条件等。

数字图像不以带有属性描述的结构化方式呈现，相反，它们会被编码为不同通道中的颜色数量，比如黑–白与红–绿–蓝通道。学习的目标是识别与特定对象相关联的模式。传统的图像识别方法是，将一幅图像转换为不同形式，比如识别对象的角点、边缘、同色斑点与基本形状。然后使用这些模式训练学习器，使之能够区分不同对象。下面列出了一些有名的传统图像识别算法。

- 边缘检测：查找一幅图像中对象的边界。
- 角点检测：识别两条边的交叉点或者其他感兴趣的点，比如行尾结束符号、曲率极大值/极小值等。

- ❑ 斑点检测：识别与周边区域有不同特征的区域，比如亮度、颜色。
- ❑ 岭检测：使用平滑函数识别图像中的兴趣点。
- ❑ **尺度不变特征变换（SIFT）**：这个算法十分强大，即使目标对象大小或方向与比对数据库中的典型样本不同，它依然能够匹配目标对象。
- ❑ 霍夫变换（Hough transform）：识别图像中的特定模式。

目前，图像识别使用的最新方法是深度学习技术。深度学习是神经网络的一种，它模仿了大脑处理信息的方法。深度学习的主要优点是，我们可以设计神经网络自动提取相关模式，这些模式反过来用于训练学习器。随着神经网络技术最新取得进展，图像识别精度得到了明显提升。比如，**ImageNet**挑战赛（ImageNet，2016）中，主办方提供了120万张图像，这些图像分别来自1000个不同分类，最佳算法的错误率由28%（2010年，利用SVM）降低到7%（2014年，利用深度神经网络）。

本章将简单了解神经网络，从最基本的构建块——**感知器**开始，逐渐引入更复杂的结构。

## 神经网络

神经网络最早出现在20世纪60年代，其灵感来自生物神经网络的研究。神经网络最新研究成果表明，深度神经网络非常适合用于模式识别任务，因为它们能够自动提取有趣特征，并且学习底层表示。这部分内容中，我们将学习从单个感知器到深度网络的基本结构与组件。

### 1. 感知器

感知器是神经网络最基本的构建单元，也是最早的监督算法之一。它定义为，用权值对输入进行加权并加上偏置。求和函数称为“**和传递函数**”（**sum transfer function**），它被送到一个**激活函数**（**activation function**）。如果激活函数到达阈值，输出为1，否则为0。这就为我们提供了一个二元分类器。感知器神经元模型如图8-1所示。

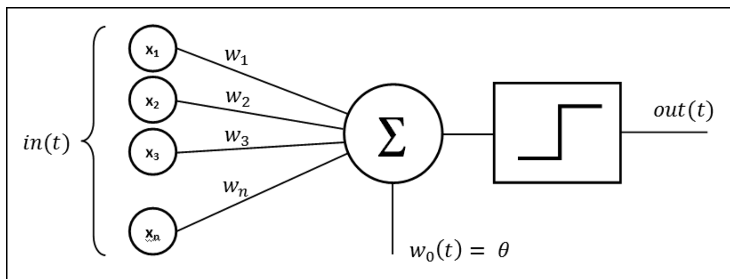



图8-1 感知器神经元模型

训练感知器使用的学习算法相当简单：先在计算输出值与正确的训练输出值之间计算误差，然后根据误差调整权重，从而实现某种形式的梯度下降算法。这个算法通常称为**delta规则**。

单层感知器不是很先进，非线性可分函数（比如XOR）不能用它建模。为了解决这个问题，人们引入了多个感知器结构，称为**多层感知器**，也叫**前馈神经网络**。

### 2. 前馈神经网络

前馈神经网络是由多个感知器组成的人工神经网络，这些感知器按层组织，可分为：输入层、输出层、一个或多个隐藏层，如图8-2所示。每层感知器（也叫神经元）与下层感知器直接相连，两个神经元之间的连接带有一个权重，类似于感知器权重。图8-2显示的是一个带有四元**输入层**的网络（对应于长度为4的特征向量）、四元**隐藏层**，以及二元**输出层**，每元对应于一个类值。

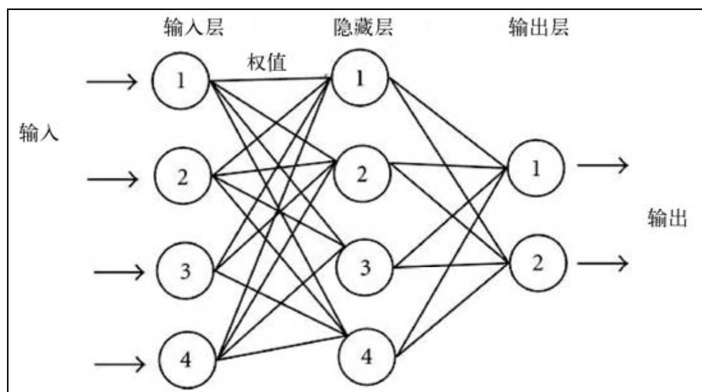



图8-2　前馈神经网络

训练多层网络最流行的方法是反向传播算法。这种算法中，采用与delta规则一样的方式，将计算得到的输出值与实际输出值进行比较。然后借助各种技术，通过网络反馈误差，调整每个连接的权重，以便减小误差值。这个过程不断重复，达到足够多的训练周期，直到错误少于某个特定阈值。

前馈神经网络可以拥有一个以上的隐藏层，每一个新增隐藏层在先前层之上创建一个新的抽象。这样做通常可以让模型变得更精确，但增加隐藏层数目会导致如下两个问题。

- **消失的梯度问题**：随着隐藏层的增多，将信息传递到先前层时，反向传播训练方法变得越来越无用，这会导致先前层训练很慢。
- **过拟合**：模型对训练数据拟合过度，对于真实样本表现糟糕。

接下来，了解用来处理这些问题的其他网络结构。

### 3. 自动编码器

自动编码器（Autoencoder）是一种前馈神经网络，其目标是学习如何压缩原数据集。我们不是将特征映射到输入层以及将标签映射到输出层，而是将特征同时映射到输入与输出层。隐藏层的元数与输入层的元数通常是不同的，这会强制网络要么扩展，要么减少原特征的数量。借助这

种方式，网络会学习那些重要特征，进而有效进行维数缩减。

图8-3是一个自动编码器的例子。如图所示，首先三元输入层扩展为四元层，然后压缩成一元层。在网络的另一侧将一元层恢复为四元层，然后再恢复为原来的三元输入层。

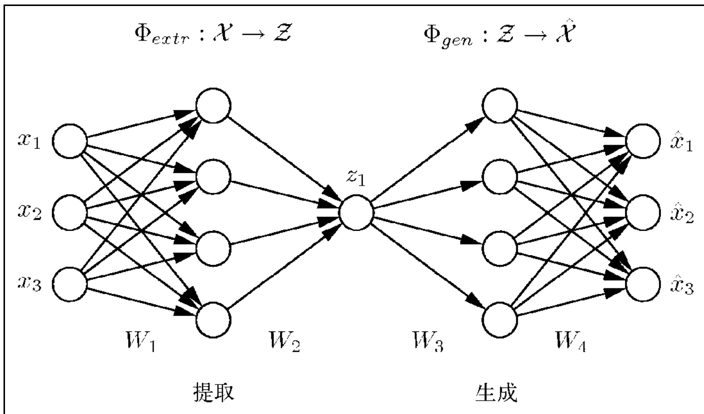



图8-3　自动编码器

一旦网络训练好之后，我们就可以利用左侧网络提取图像特征，就像我们在传统图像处理中所做的那样。

还可以将多个自动编码器组成**堆叠式自动编码器**，如图8-4所示。前面已经对最基本的自动编码器做了讲解，这里要讨论其隐藏层。然后，选取学好的隐藏层（圆圈），并且重复这个步骤，学习更多的抽象表示。可以多次重复这个过程，将原特征转换为越来越少的维数。最后，选择所有隐藏层，将其堆叠为一个规则的前馈网络，如图8-4右上角所示。

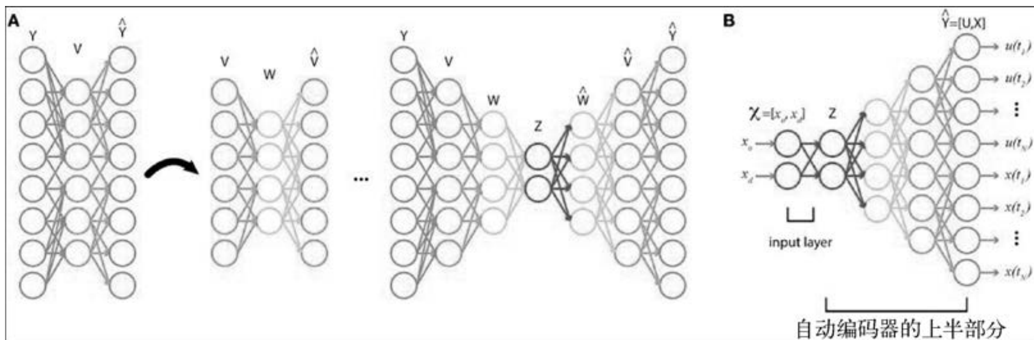



图8-4　堆叠式自动编码器

#### 4. 受限玻尔兹曼机

受限玻尔兹曼机（Restricted Boltzman machine，RBM）是一种无向神经网络，也称为**生成式**

**随机神经网络**（**Generative Stochastic Networks，GSN**），它能够在输入集之上学习概率分布。顾名思义，它起源于玻尔兹曼机（Boltzman machine），这是一种20世纪80年代出现的循环神经网络。“受限”是指神经元必须组成两个全连接层——输入层与隐藏层，如图8-5所示。

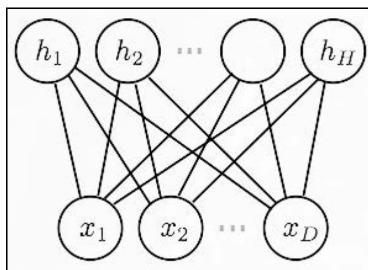



图8-5　受限玻尔兹曼机

不同于前馈网络，可见层与隐藏层之间的连接是无向的。因此，值可以沿着“可见–隐藏”与“隐藏–可见”方向传播。

受限玻尔兹曼机的训练基于**对比散度算法**（**Contrastive Divergence**），使用类似反向传播的梯度下降过程更新权重，将**吉布斯采样**（**Gibbs sampling**）应用到**马尔可夫链**以评估梯度——权重的改变方向。

我们也可以堆叠受限玻尔兹曼机，形成**深度信念网络**（**Deep Belief Networks，DBN**）。此情形之下，RBM的隐藏层充当RBM层的可见层，如图8-6所示。

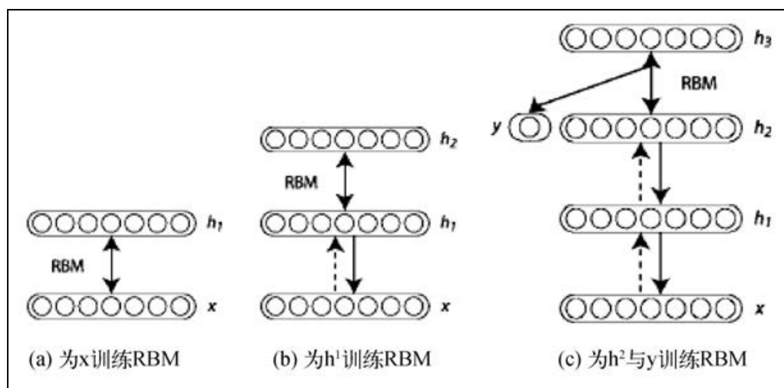



图8-6　深度信念网络

训练是渐进式的，即逐层训练。

5. 深度卷积网络

最近，在图像识别测试中取得很好效果的一种网络结构是**卷积神经网络**（**Convolutional**

**Neural Network，CNN**）。它是前馈神经网络的一种，模拟视觉皮层的行为，用于探索输入图像的2D结构，即展现空间局部相关性的模式。

CNN网络由若干卷积与子采样层组成，后面可以有全连接层。图8-7显示的是一个CNN网络。输入层读取一幅图像中的所有像素，然后应用多个过滤器。图中应用了4个不同的过滤器。每个过滤器都应用到原图像，比如一个6×6过滤器的一个像素被计算为输入像素的6×6平方与相应的6×6权重之和。这实际引入了与处理标准图像类似的过滤器，比如平滑、相关、边缘检测等。这样产生的结果图像称为**特征图**（**feature map**）。图像例子中，我们有4个特征图，每一个对应一个过滤器。

接下来的层是子采样层，它用于减少输入大小。在2×2的连续区域上（大图像高达5×5），通常采用平均值或最大池化（max pooling）方法，对每个特征图做子采样。比如，如果特征图大小是16×16，子采样区域是2×2，缩减后的特征图尺寸是8×8，通过计算最大、最小、平均值或者用其他函数，将4个像素（2×2方格）合并成一个像素。

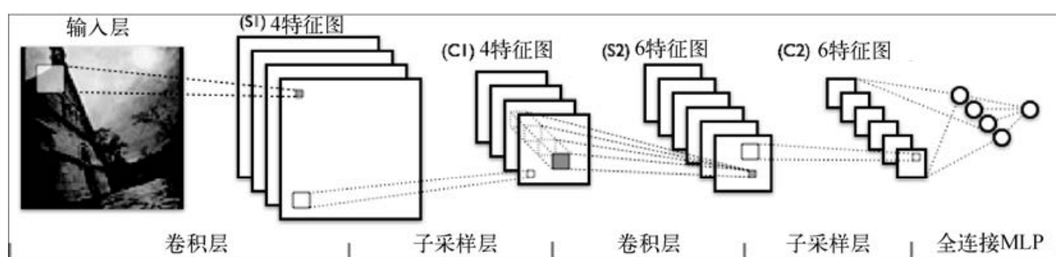



图8-7　CNN网络

网络可以包含几个连续卷积与子采样层，如图8-7所示。一个特定的特征图会被连接到下一个缩减/卷积特征图，而相同层上的特征图不会彼此相连。

在最后的子采样层或卷积层之后，通常会有一个全连接层（与标准多层神经网络中的层完全相同），表示目标数据。

CNN训练采用修改过的反向传播算法，它会把子采样层也一起考虑进来，并且基于所有应用过滤器的值更新卷积过滤器的权重。



在ImageNet比赛结果页面可以看到一些好的CNN设计：

http://www.image-net.org/

A. Krizhevsky等人撰写的ImageNet Classification with Deep Covolutional Neural Networks论文中介绍了AlexNet。

至此，我们大致了解了主要的神经网络结构。接下来，我们将学习如何实际实现。

## 8.2　图像分类

本节将讨论如何使用Deeplearning4j库实现一些神经网络结构。

### 8.2.1　Deeplearning4j

第2章已经提到过Deeplearning4j库，它是Java与Scala环境下的开源分布式深度学习项目。Deeplearning4j依赖Spark与Hadoop使用MapReduce框架，并行训练模型，且反复对中心模型中产生参数进行平均。关于这个库的详细介绍，请参考第2章相关内容。

#### 获取DL4J

获取Deeplearning4j最简便的方法是通过Maven仓库：

- 新建一个Eclipse项目，选择**Maven Project**，如图8-8所示。

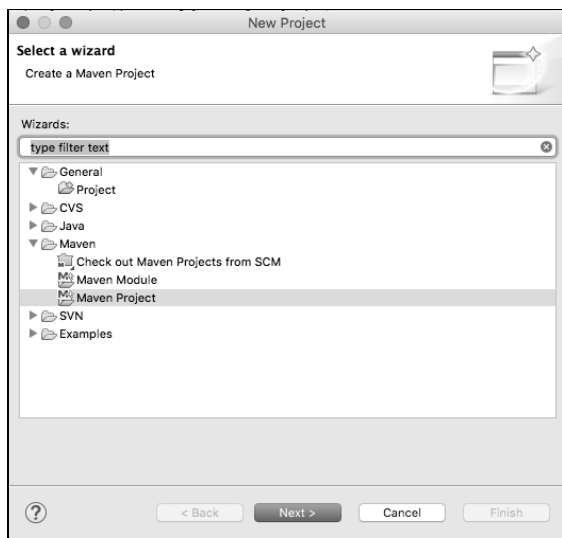



图8-8　选择Maven Project

- 打开pom.xml文件，在<dependencies>部分添加如下依赖关系：

```
<dependency>
    <groupId>org.deeplearning4j</groupId>
    <artifactId>deeplearning4j-nlp</artifactId>
    <version>${dl4j.version}</version>
</dependency>

<dependency>
    <groupId>org.deeplearning4j</groupId>
```

```
        <artifactId>deeplearning4j-core</artifactId>
        <version>${dl4j.version}</version>
    </dependency>
```

- 最后，右键单击**Project**，选择**Maven**，选择**Update project**。

## 8.2.2　MNIST 数据集

MNIST数据集是最著名的数据集之一，由手写数字组成，如图8-9所示。该数据集包含60 000个训练与10 000个测试图像。

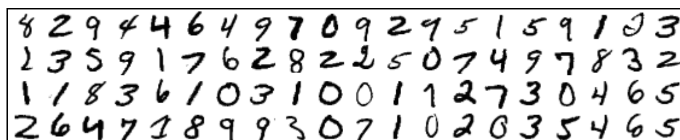


图8-9　MNIST数据集

这个数据集通常用在图像识别问题中，以测试算法性能。最差记录的错误率是12%，测试时使用单层神经网络中的SVM算法，并且没有做预处理。截止到2016年，最低的错误率只有0.21%，使用的是**DropConnect**神经网络；紧随其后的是**深度卷积网络**，错误率为0.23%；然后是深度前馈网络，错误率是0.35%。

接下来，让我们看看如何加载数据集。

## 8.2.3　加载数据

Deeplearning4j 提供了“开箱即用”的 MNIST 数据集加载器。加载器被初始化为`DataSetIterator`。先导入`DataSetIterator`类与所有支持的数据集，这些数据集是`impl`包的一部分，包含的数据集有Iris、MNIST及其他。

```
import org.deeplearning4j.datasets.iterator.DataSetIterator;
import org.deeplearning4j.datasets.iterator.impl.*;
```

接着定义一些常量，比如28 × 28个像素组成的图像，有10个目标类与60 000个样本。新初始化一个`MnistDataSetIterator`类，用于下载数据集及其标签。参数分别是迭代批大小、总样本数，以及是否将数据集二值化：

```
int numRows = 28;
int numColumns = 28;
int outputNum = 10;
int numSamples = 60000;
int batchSize = 100;
DataSetIterator iter = new MnistDataSetIterator(batchSize, numSamples,true);
```

8

有一个已经实现的数据导入器真的很方便，但它对你自己的数据不起作用。下面快速了解一下它是如何实现的，以及如何调整才能让它支持你的数据。如果你想迫不及待地开始实现神经网络，那么完全可以跳过下面这部分内容。当你需要导入自己的数据时，再回过头来学习。



为了加载我们自己的数据，你需要实现两个类，一个是DataSetIterator类，保存数据集的所有信息；另一个是BaseDataFetcher类，用于实际从文件、数据库或Web拉取数据。在GitHub上你可以看到这两个类的示例实现：https://github.com/deeplearning4j/deeplearning4j/tree/master/deeplearning4j-core/src/main/java/org/deeplearning4j/datasets/iterator/impl。

另一个选择是使用Canova库，它由同一个作者开发：http://deeplearning4j.org/canovadoc/。

## 8.2.4　创建模型

本节将讨论如何实际创建一个神经网络模型。先创建一个基本的单层神经网络建立一个基准标杆，并且学习基本操作。随后，使用DBN与**多层卷积网络**改进初始结果。

### 1. 创建单层回归模型

先创建一个单层回归模型，它基于softmax激活函数，如图8-10所示。由于我们只有一个层，所以神经网络的**输入**是所有图形像素，即28×28=**784**个神经元。**输出**神经元的个数为**10**，对应于每一个数字。网络中的层是全连接的，如图8-10所示。

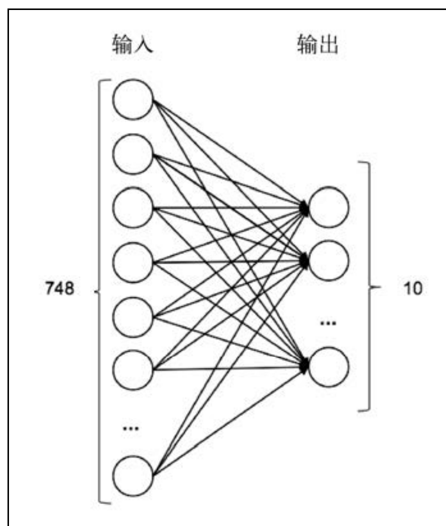



图8-10　单层回归模型

使用一个NeuralNetConfiguration Builder对象定义神经网络，如下：

```
MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder();
```

接着，为梯度搜索定义参数，以便使用**共轭梯度最优化**算法做迭代。其中，momentum参数指定优化算法收敛到局部最优的速度，momentum值越高，训练得越快；但速度过快有可能降低模型准确度，代码如下：

```
.seed(seed)
.gradientNormalization(GradientNormalization.ClipElementWiseAbsolu
  teValue)
   .gradientNormalizationThreshold(1.0)
   .iterations(iterations)
   .momentum(0.5)
   .momentumAfter(Collections.singletonMap(3, 0.9))
   .optimizationAlgo(OptimizationAlgorithm.CONJUGATE_GRADIENT)
```

接下来，指定网络有一个层，并且定义错误函数（**NEGATIVELOGLIKELIHOOD**）、内部感知器激活函数（**softmax**），以及输入和输出层的数量，对应于总的图像像素和目标变量数：

```
.list(1)
.layer(0, new
OutputLayer.Builder(LossFunction.NEGATIVELOGLIKELIHOOD)
.activation("softmax")
.nIn(numRows*numColumns).nOut(outputNum).build())
```

最后，为网络开启预训练（**pretrain**），关闭反向传播，实际创建未经训练的网络结构：

```
   .pretrain(true).backprop(false)
   .build();
```

一旦网络结构定义完成后，可以用它初始化一个**MultiLayerNetwork**对象，代码如下：

```
MultiLayerNetwork model = new MultiLayerNetwork(conf);
model.init();
```



接下来，调用**setListeners**方法，绑定模型与训练数据，代码如下：

```
model.setListeners(Collections.singletonList((IterationListener)
  new ScoreIterationListener(listenerFreq)));
```

此外，调用**fit(int)**方法触发端对端的网络训练：

```
model.fit(iter);
```

为了评价模型，新建并初始化一个**Evaluation**对象，用于存储批结果：

```
Evaluation eval = new Evaluation(outputNum);
```

然后，在数据集上分批做迭代，以便让内存消耗保持在一个合理的范围内，并且结果保存在一个**eval**对象中：

```
DataSetIterator testIter = new MnistDataSetIterator(100,10000);
while(testIter.hasNext()) {
    DataSet testMnist = testIter.next();
    INDArray predict2 =
    model.output(testMnist.getFeatureMatrix());
    eval.eval(testMnist.getLabels(), predict2);
}
```

最后，调用`stats()`函数获取结果：

```
log.info(eval.stats());
```

基本的单层模型准确度如下：

```
Accuracy:  0.8945
Precision: 0.8985
Recall:    0.8922
F1 Score:  0.8953
```

从上面结果可以看到，模型准确率是89.22%，错误率为10.88%，这表示模型在MNIST数据集上表现很差。接下来对模型做进一步改善，将其从简单的单层网络变为带有适度复杂度的深度信念网络，这种网络使用了受限玻尔兹曼机与多层卷积网络。

### 2. 创建深度信念网络

本节将创建一个基于受限玻尔兹曼机的深度信念网络，如图8-11所示。深度信念网络由4个层组成，第一个层将**784**个输入缩小为**500**个神经元，然后是**250**个，再然后是**200**个，最后是10个目标值：

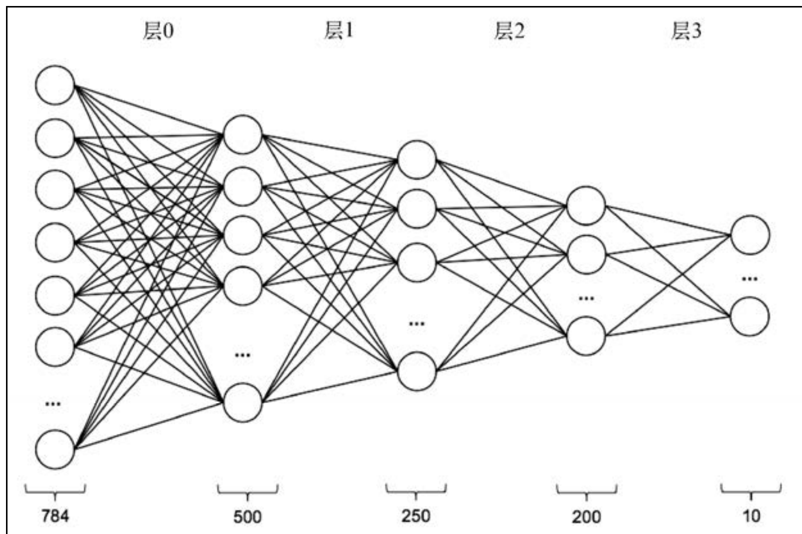



图8-11　基于受限玻尔兹曼机的深度信念网络

代码与上一个例子一样，下面看看如何配置这样一个网络：

```
MultiLayerConfiguration conf = new
  NeuralNetConfiguration.Builder()
```

接着定义梯度优化算法，代码如下：

```
.seed(seed)
.gradientNormalization(
GradientNormalization.ClipElementWiseAbsoluteValue)
.gradientNormalizationThreshold(1.0)
.iterations(iterations)
.momentum(0.5)
.momentumAfter(Collections.singletonMap(3, 0.9))
.optimizationAlgo(OptimizationAlgorithm.CONJUGATE_GRADIENT)
```

然后，指定网络有4个层：

```
.list(4)
```

第一个层的输入有784个神经元，输出为500个神经元。使用均方根误差交叉熵——Xavier算法初始化权重，基于输入与输出神经元的数目自动确定初始化权重的范围，代码如下：

```
.layer(0, new RBM.Builder()
.nIn(numRows*numColumns)
.nOut(500)
.weightInit(WeightInit.XAVIER)
.lossFunction(LossFunction.RMSE_XENT)
.visibleUnit(RBM.VisibleUnit.BINARY)
.hiddenUnit(RBM.HiddenUnit.BINARY)
.build())
```

接下来的两个层拥有相同的参数，但是输入与输出神经元的数目有所不同：

```
.layer(1, new RBM.Builder()
.nIn(500)
.nOut(250)
.weightInit(WeightInit.XAVIER)
.lossFunction(LossFunction.RMSE_XENT)
.visibleUnit(RBM.VisibleUnit.BINARY)
.hiddenUnit(RBM.HiddenUnit.BINARY)
.build())

.layer(2, new RBM.Builder()
.nIn(250)
.nOut(200)
.weightInit(WeightInit.XAVIER)
.lossFunction(LossFunction.RMSE_XENT)
.visibleUnit(RBM.VisibleUnit.BINARY)
.hiddenUnit(RBM.HiddenUnit.BINARY)
.build())
```

现在，使用最后一个层将神经元映射到输出。使用**softmax**激活函数完成这个任务，代码如下：

```
.layer(3, new OutputLayer.Builder()
.nIn(200)
.nOut(outputNum)
.lossFunction(LossFunction.NEGATIVELOGLIKELIHOOD)
.activation("softmax")
.build())
.pretrain(true)
.backprop(false)
.build();
```

训练与评价的其他部分与单层网络的例子是一样的。请注意，训练深度网络耗费的时间可能明显要比训练单层网络长得多，但准确度应该能够达到93%左右。

接下来，让我们看一看另外一个深度网络。

#### 3. 创建多层卷积网络

本章最后一个例子中，让我们一起学习如何创建卷积网络，如图8-12所示。这个卷积网络由7个层组成：首先，用max pooling重复两对卷积与子采样层；然后将最后一个子采样层连接到紧密相连的前馈神经网络，最后三个层中依次含有120个神经元、84个神经元、10个神经元。这样一种网络实际上组成了完整的图像识别管道，前4个图层对应于特征提取，后3个图层对应于学习模型：

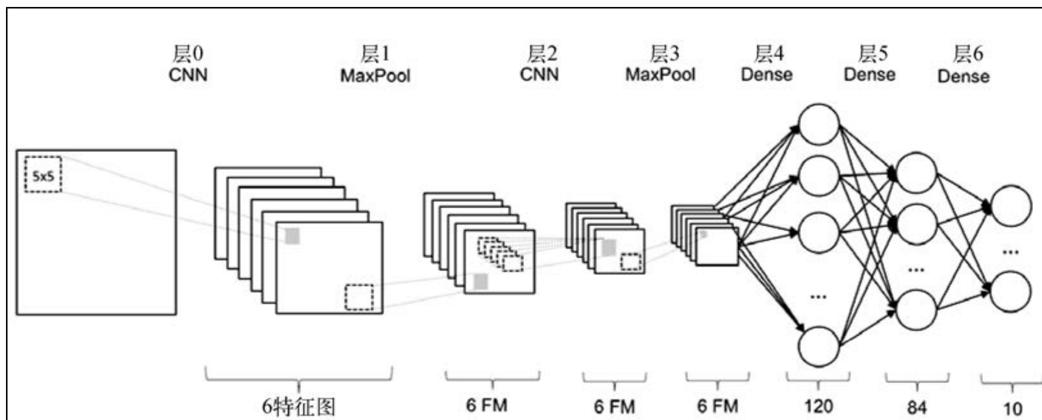



图8-12　创建卷积网络

网络配置的初始化方法与前面一样，如下：

```
MultiLayerConfiguration.Builder conf = new
  NeuralNetConfiguration.Builder()
```

接着，指定梯度下降算法及其参数，代码如下：

```
.seed(seed)
.iterations(iterations)
```

```
.activation("sigmoid")
.weightInit(WeightInit.DISTRIBUTION)
.dist(new NormalDistribution(0.0, 0.01))
.learningRate(1e-3)
.learningRateScoreBasedDecayRate(1e-1)
.optimizationAlgo(
OptimizationAlgorithm.STOCHASTIC_GRADIENT_DESCENT)
```

此外，还要指定7个网络层，如下：

```
.list(7)
```

第一个卷积层的输入是一幅完整图像，而输出是6个特征图。卷积层应用一个5 × 5的过滤器，结果存储在1 × 1单元格中：

```
.layer(0, new ConvolutionLayer.Builder(
    new int[]{5, 5}, new int[]{1, 1})
    .name("cnn1")
    .nIn(numRows*numColumns)
    .nOut(6)
    .build())
```

第二个层是子采样层，它接收一个2 × 2区域，并把最大结果存成2 × 2元素：

```
.layer(1, new SubsamplingLayer.Builder(
SubsamplingLayer.PoolingType.MAX,
new int[]{2, 2}, new int[]{2, 2})
.name("maxpool1")
.build())
```

接下来的两个层重复前面两个层：

```
.layer(2, new ConvolutionLayer.Builder(new int[]{5, 5}, new
  int[]{1, 1})
    .name("cnn2")
    .nOut(16)
    .biasInit(1)
    .build())
.layer(3, new SubsamplingLayer.Builder
  (SubsamplingLayer.PoolingType.MAX, new
  int[]{2, 2}, new int[]{2, 2})
    .name("maxpool2")
    .build())
```

接着，将子采样层的输出连接到稠密前馈网络，先是**120**个神经元，然后穿过另一个层，变成**84**个神经元，代码如下：

```
.layer(4, new DenseLayer.Builder()
    .name("ffn1")
    .nOut(120)
    .build())
.layer(5, new DenseLayer.Builder()
    .name("ffn2")
```

```
    .nOut(84)
    .build())
```

最后一个层将**84**个神经元与**10**个输出神经元连接在一起：

```
.layer(6, new OutputLayer.Builder
  (LossFunctions.LossFunction.NEGATIVELOGLIKELIHOOD)
    .name("output")
    .nOut(outputNum)
    .activation("softmax") // 需要径向基函数（radial basis function）
    .build())
.backprop(true)
.pretrain(false)
.cnnInputSize(numRows,numColumns,1);
```

为了训练这个结构，可以重用前面两个例子中编写的代码。再次提醒，训练可能要花一些时间，训练结束后，网络的准确度应该能达到98%左右。



由于模型训练主要依赖线性代数运算，所以使用**GPU**（**图形处理单元**）可以明显提高训练速度。写作本书时，GPU后端正在进行重写，请访问如下网址查看最新文档：http://deeplearning4j.org/documentation。

正如在这些例子中看到的那样，越复杂的神经网络越能让我们自然地提取相关特征，因而可以完全避免传统的图像处理过程。然而，我们为此付出的代价是处理时间变长，而且还要有大量学习样本使这种方法有效。

## 8.3　小结

本章讨论了如何识别图像中的模式以区分不同的类，还介绍了深度学习的基本原理，并且学习了如何使用Deeplearning4j库进行实现。我们从基本的神经网络结构讲起，进而讲解如何实现它们，借以解决手写数字识别问题。

下一章将进一步研究模式问题，但研究的不再是图像中的模式，而将学习使用传感器数据中的时序依赖处理模式问题。

第 9 章

# 利用手机传感器进行行为识别

上一章主要讲了图像中的模式识别问题，而本章主要讲解如何识别传感器数据中的模式。相比于图像，这些模式带有时序依赖。我们将讨论如何通过手机内置的惯性传感器识别日常行为，比如步行、坐下、跑步等。此外，本章还将提供一些有关研究的参考，着重介绍行为识别社区中的一些最佳实践。

本章内容涵盖如下主题：

- 行为识别简介，包括手机传感器与行为识别流水线
- 从移动设备收集传感器数据
- 讨论行为分类与模型评价
- 部署行为识别模型

## 9.1　行为识别简介

行为识别是行为分析中一个基础步骤，广泛应用于健康生活、健身跟踪、远程协助、安保应用、老年人护理等领域。从加速度传感器、陀螺仪传感器、压力传感器、GPS等传感器获取的都是低级数据，行为识别要将这些数据转换为高级的行为原语（behavior primitives）描述。大多数情况下，有些行为是基本的，比如步行、坐卧、跳跃等，如图9-1所示；还有些行为比较复杂，比如去工作、准备早饭、购物等。

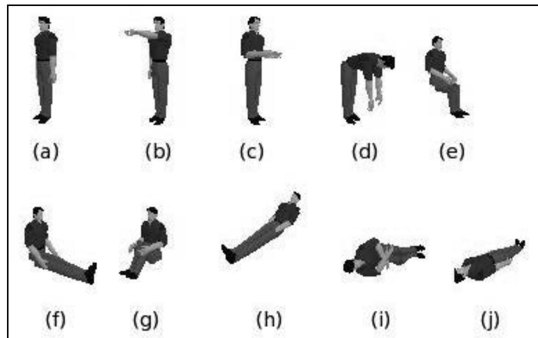



图9-1　基本行为

本章将讨论如何向移动应用添加行为识别功能。首先，先了解行为识别问题是什么、需要收集哪类数据、面临的主要困难以及如何解决。

然后，通过一个例子向大家演示如何在一个Android应用中实现行为识别功能，包括数据收集、数据转换以及创建分类器。

让我们开始吧！

## 9.1.1　手机传感器

先了解一下手机传感器有哪些，以及它们能提供什么。目前，大部分智能设备都内置了一些传感器，用于感知运动、位置、方向以及周围环境条件。由于传感器能够提供高精确度、高频率、高准确度的数据，所以可以使用这些数据重建用户复杂的移动、手势、运动等动作。各种应用经常会用到传感器，比如陀螺仪数据可以控制游戏中的某个对象，GPS数据可以定位用户，加速度传感器数据推断用户正在做的动作，如骑自行车、跑步或步行。

图9-2的几个例子展示了用户通过这些传感器可以做的交互动作。

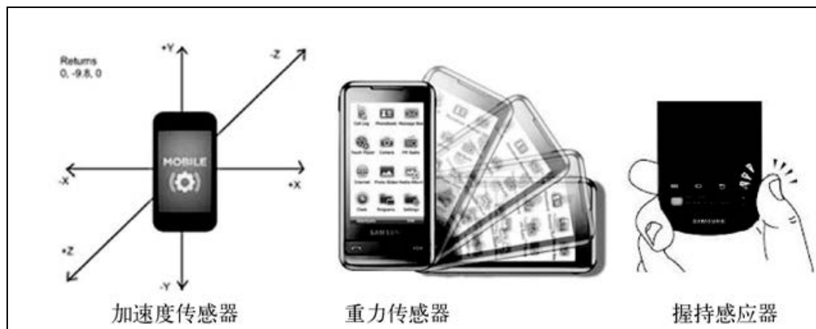



图9-2　用户通过手机传感器可以做的交互动作

手机中内置的传感器大致可划分为如下3类。

- **运动传感器**：沿着3个相互垂直的坐标轴测量加速度与旋转力。这类传感器包括加速度传感器、重力传感器与陀螺仪传感器。
- **环境传感器**：用于测量各种环境参数，比如照明、温度、压力、湿度。这类传感器有气压计、光度计、温度计。
- **位置传感器**：用于测量设备的物理位置。这类传感器有方向传感器与磁力计。



有关不同移动平台更详细的描述，请访问如下页面。

- Android传感器框架：http://developer.android.com/guide/topics/sensors/sensors_overview. html
- iOS Core运动框架：https://developer.apple.com/library/ios/documentation/CoreMotion/Reference/CoreMotion_Reference/
- Windows Phone：https://msdn.microsoft.com/en-us/library/windows/apps/hh202968(v=vs.105). aspx

本章将只用到Android传感器框架。

### 9.1.2 行为识别流水线

前面讲解了对称名数据做分类的内容，与之相比，对多维时序传感器数据做分类本身要复杂得多。首先，从时间上看，每次观测都与上一次和下一次观测相连，这使得我们很难只对单个观测集做简单分类。其次，在不同时间点从传感器随机获取的数据是不可预测的，因为它们可能受到传感器噪声、环境干扰或其他各种因素影响。而且，一个行为可由多个子行为组成，每个子行为以不同方式做出，每个人所做的行为也会有不同，这会产生很高的组内差异。最后，所有这些原因会让一个行为识别模型变得不准确，经常将新数据分错类。对于一个行为识别分类器，要求的特性之一是它要在识别的行为序列中保持连续性和一致性。

为了应对这些挑战，将行为识别应用于图9-3所示流水线。

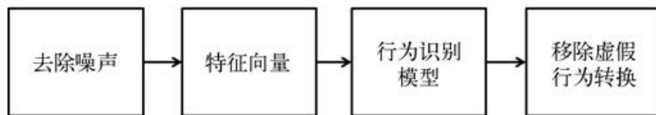



图9-3 行为识别应用流水线

在第一步中，尽量去除数据中的噪声，常用的方法有降低传感器采样率、移除异常值、使用高通/低通滤波器等。接下来，构建一个特征向量，例如，通过使用**离散傅里叶变换**（**DFT**）方法，将传感器数据从时间域转换为频率域。DFT方法接收一系列样本输入，然后返回一系列按照

频率排列的正弦系数，它们表示源采样中出现的频率的组合。



关于傅里叶变换，Pete Bevelacqua写了一篇不错的介绍文章，请前往http://www.thefouriertrans-form.com阅读。

如果想了解更多有关傅里叶变换的技术与理论背景，请观看Robert Gallanger与Lizhong Zheng两人的MIT公开课的第8讲与第9讲：

http://theopenacademy.com/content/principlesdigital-communication

接下来，基于特征向量与训练数据集，我们可以创建一个行为识别模型，它把一个原子操作赋予每次观测。因此，对于每次新的传感器读数，模型会输出最有可能的行为标签。然而，模型会犯错。故而在最后一步，通过移除现实中不会出现的转换，在行为之间做平滑转换。比如，在少于半秒的时间内，在"躺着–站起–躺着"动作之间完成转换是不可能的，所以把动作之间这样的转换平滑为"躺着–躺着–躺着"。

创建行为识别模型时使用的是监督学习方法，它由训练与分类步骤组成。训练步骤中，使用一组带有标签的数据训练模型。第二步是使用训练过的模型为新的、未见过的数据指派标签。这两个阶段中，数据必须使用同一套工具做预处理，比如过滤与特征向量计算。

后处理阶段（移除虚假行为）本身也可能是一个模型，因此也需要进行学习。在此情形下，预处理步骤还包括行为识别，这让分类器的安排变成了一个元学习问题。为了避免过拟合，最重要的是让训练后处理阶段时使用的数据集与训练行为识别模型所用的数据集不一样。

我们将大致跟着Andrew T. Campbell教授所讲的智能手机编程课程来做，他来自达特茅斯大学。在课程中，他开发了一款收集数据的移动App（Campbell，2011），后面会用到它。

### 9.1.3　计划

计划由训练阶段与部署阶段组成。训练阶段可归结为如下步骤：

- 安装Android Studio，导入`MyRunsDataCollector.zip`。
- 在你的Android手机中加载应用。
- 收集你的数据，比如站立、步行、跑步，把数据转换为一个特征向量，包括FFT变换。不必惊慌，我们无需编写FFT这类低级信号处理函数，因为会使用现成代码去做这个工作。数据会保存到你的手机中名为`features.arff`的文件。
- 使用导入的数据，创建并评价一个行为识别分类器，并且实现移除虚假行为转换的过滤器。
- 将分类器放回移动应用。

如果没有Android手机，或者想跳过所有与移动应用相关的步骤，可以使用一个已经收集好的数据集，它存在于`data/features.arff`文件中，然后直接学习9.3节的内容。

## 9.2　从手机收集数据

下面这部分内容涉及规划的前3个步骤。如果你想直接使用现成的数据，可以跳过本节内容，直接学习9.3节。收集传感器数据的开源移动应用有很多，Prof.Campbell编写的那个App就是其中之一，本章将使用它收集数据。这个应用为不同行为类型实现了收集传感器数据的功能，比如站立、步行、跑步等。

先准备Android开发环境。如果已经安装，可以直接跳到9.2.2节。

### 9.2.1　安装 Android Studio

Android Studio是一个Android平台开发环境。详细安装步骤不再赘述，在手机启动App所需的基本设置保持默认值不变即可。关于Android开发环境的详细内容，建议阅读Kyle Mew编写的*Android 5 Programming by Example*一书。

首先，从http://developer.android.com/sdk/installing/index.html?pkg=studio下载最新的Android Studio开发者版本，然后根据提示进行安装。整个安装过程超过10分钟，所需空间大约0.5 GB。

安装过程如图9-4所示。

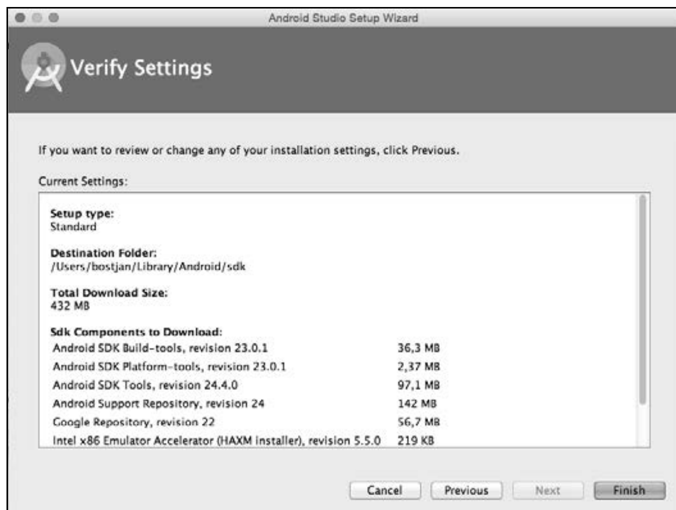



图9-4　安装Android Studio开发者版本

### 9.2.2　加载数据采集器

首先，从http://www.cs.dartmouth.edu/~campbell/cs65/code/myrunsdatacollector.zip下载`MyRuns-`

DataCollector源代码。安装好Android Studio之后，选择Open an existing Android Studio project，如图9-5所示。选择MyRunsDataCollector文件夹，将项目导入Android Studio。

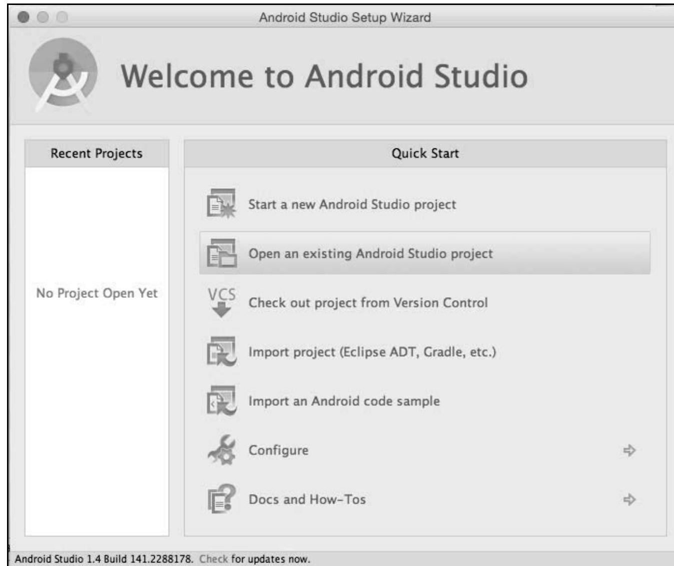



图9-5　将项目导入Android Studio

项目导入完成后，应该能够看到项目文件结构，如图9-6所示。如你所见，数据采集器由CollectorActivity.java、Globals.java与SensorsService.java组成。项目中也包含一个FFT.java文件，它用来进行低级信号处理。

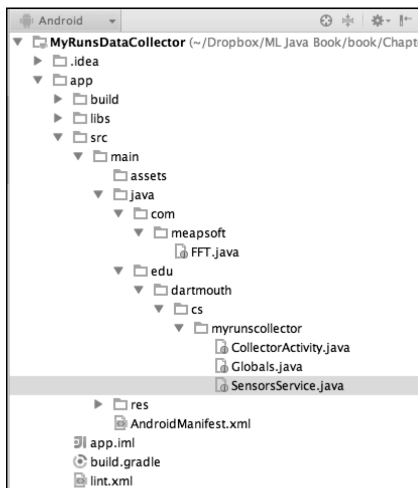



图9-6　项目文件结构

myrunscollector是最主要的包，包含如下类。

- Globals.java：定义全局常量，比如行为标签、ID、数据文件名等。
- CollectorActivity.java：该类实现了用户界面操作，即按下特定按钮时会发生什么。
- SensorsService.java：该类实现的服务用来收集数据、计算特征向量（稍后讲解），以及将数据存储到手机文件。

接下来要处理的是如何设计特征。

**特征提取**

"为人类行为找到合适的表示"可能是行为识别中最具挑战的部分。我们要使用简单且一般的特征表示行为，这样使用这些特征的模型才具有一般性，才能在学习集中很好地区分一些行为与另一些行为。

事实上，为训练集中特定的观测结果设计特征并不难，并且这些特征也会有很好的工作表现。但是，由于训练集抓取的只是整个人类行为的一部分，所以过于具体的特征在一般行为上可能会失效。

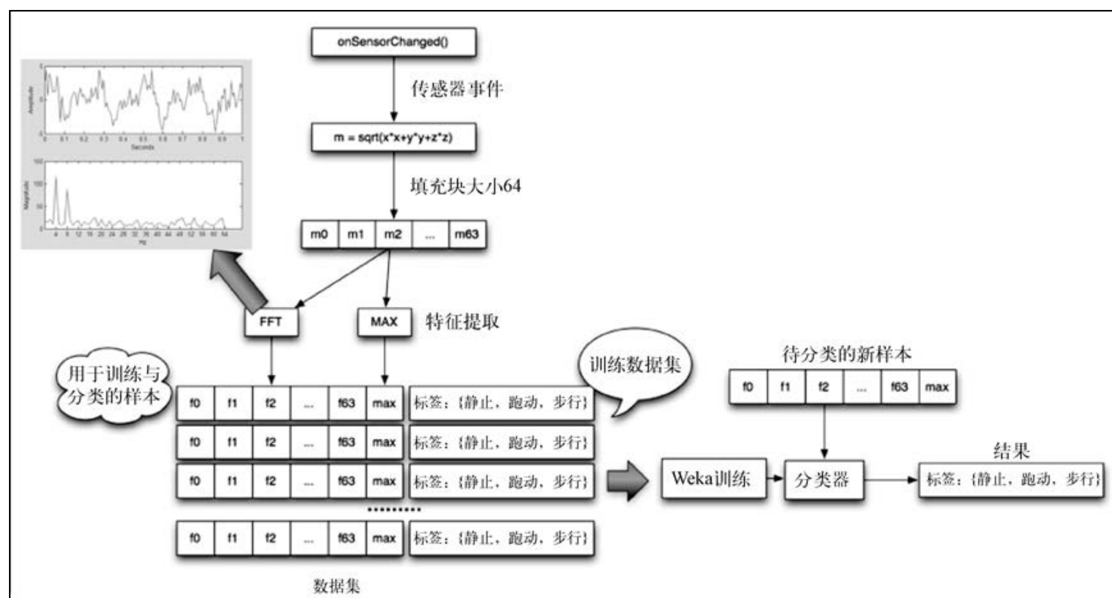



图9-7　特征提取

下面看看这在MyRunsDataCollector类中是如何实现的。应用程序启动时，onSensorChanged()方法会从加速度传感器得到3个带有特定时间戳的读数（$x$、$y$、$z$），并且根据传感器读数计算量级。计算FFT系数之前，这些方法缓存高达64个连续量级（Campbell，2015）：

“如左上图所示，FFT将随时间变化的振幅的时间序列转换为频率大小（振幅的某种表示）；例子显示的是某个振荡系统，主频率在4~8周/秒（称为赫兹，H）（想象有一个系在橡皮筋上的球在短时间内拉伸、振荡，或者你在步行、跑步时的步态）——可能有人见过这样的时间域与频率域系统。*x*、*y*、*z*加速度器读数和大小是时间域变量。我们将这些时间域数据转换为频率域，因为这能产生紧凑的分布，分类器会使用它创建决策树模型。比如，转置为频率域的振幅率看上去可能像图底部的散点，顶部散点是时间域，底部散点是把时间域转换为频率域的样子。

训练阶段也会使用采集器存储(m0…m63)的最大（MAX）量级与用户提供的标签（比如步行）。个别特征会被计算为量级（f0…f63）、MAX量级和类标签。”

下面开始收集实际数据。

### 9.2.3　收集训练数据

现在，可以使用采集器为行为识别收集训练数据。采集器默认支持3种行为：站立、步行、跑步，在下面应用程序的屏幕截图中可以看到它们。

你可以先选择一个行为，即目标类值，然后点击**START COLLECTING**按钮开始记录数据。收集数据时，请确保每个行为记录时间不少于3分钟。比如选择**Walking**行为，点击**START COLLECTING**按钮之后，要在周围走动不少于3分钟。等到3分钟后，点击**STOP COLLECTING**按钮停止收集数据。针对每个行为做数据收集时，只要重复这个过程即可。

此外，你还可以收集包含这些行为的不同场景，比如在厨房中走动、在户外走动、沿直线行走等。这样，你将拥有每个行为类的更多数据，以及更好的分类器。这很有道理，不是吗？数据越多，分类器就越少迷茫。如果你只有少量数据，就会发生过拟合问题，分类器就会犯迷糊，分不清行走中的站立、小跑式的行走。数据越多，它们就越不会迷糊。进行调试时，为每个行为收集数据的时间可能少于3分钟。但对于最终成品，数据越多，性能越好。只要在相同文件中简单积累多个记录实例即可。

请注意，点击DELETE DATA按钮后，存储在手机文件中的数据会被删除。如果想重新开始，请先点击DELETE DATA按钮，否则新采集的数据会被添加到文件末尾。

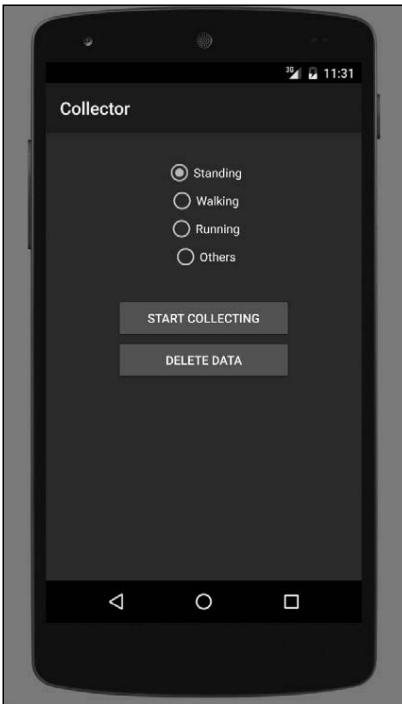



图9-8　点击DELETE DATA按钮

采集器实现了我们前面提到的示意图：它收集加速度计数据样本、计算大小、使用`FFT.java`类计算系数，并且生成特征向量。然后，将数据存储到Weka格式的`features.arff`文件。根据收集的数据量的大小，特征向量的数目不一样。收集数据的时间越长，累积的特征向量越多。

停止使用采集器工具收集训练数据之后，需要抓取数据让工作流继续。可以使用**Android Device Monitor**中的文件浏览器，将`features.arff`文件从手机上传到电脑进行保存。点击菜单右侧的Android机器人小图标，可以访问Android Device Monitor，如图9-9所示。

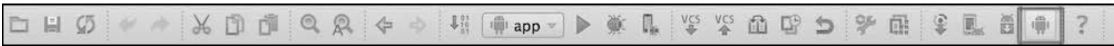



图9-9　访问Android Device Monitor

在左侧窗格中选择你的设备，手机中存储的内容就会显示在右侧。导航到`mnt/shell/emulated/Android/data/edu.dartmouth.cs.myrunscollector/files/features.arff`，如图9-10所示。

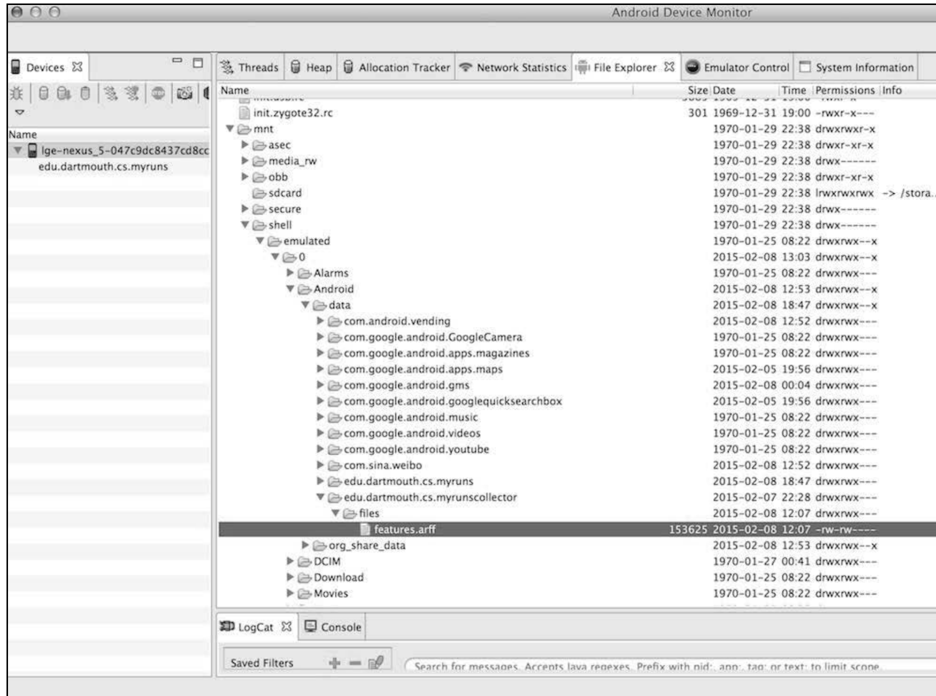


图9-10　选择设备并导航

为了将`features.arff`文件上传到你的计算机，需要先选择该文件（高亮显示），然后点击**Upload**按钮。

下面创建分类器。

## 9.3　创建分类器

一旦将传感器数据样本表示为带有类别指派的特征向量，我们就可以应用常规技术监督分类，包括特征选择、特征离散化、模型学习、*k*折交叉验证等。本章不会深入探究机器学习算法的细节。可以应用任何一种支持数字特征（numerical features）的算法，包括SVM、随机森林、AdaBoost、决策树、神经网络、多层感知器等。

先从简单的决策树开始：加载数据集、创建类别属性、创建决策树模型、输出模型。

```
String databasePath = "/Users/bostjan/Dropbox/ML Java Book/book/
datasets/chap9/features.arff";

//加载arff格式数据
Instances data = new Instances(new BufferedReader(new FileReader(databasePath)));
```

```
//设置class的最后属性为类别
data.setClassIndex(data.numAttributes() - 1);

//创建基本的决策树模型
String[] options = new String[]{};
J48 model = new J48();
model.setOptions(options);
model.buildClassifier(data);

//输出决策树
System.out.println("Decision tree model:\n"+model);
```

算法输出模型如下：

```
Decision tree model:
J48 pruned tree
------------------

max <= 10.353474
|   fft_coef_0000 <= 38.193106: standing (46.0)
|   fft_coef_0000 > 38.193106
|   |   fft_coef_0012 <= 1.817792: walking (77.0/1.0)
|   |   fft_coef_0012 > 1.817792
|   |   |   max <= 4.573082: running (4.0/1.0)
|   |   |   max > 4.573082: walking (24.0/2.0)
max > 10.353474: running (93.0)

Number of Leaves  : 5

Size of the tree : 9
```

这个决策树相当简单，看上去也很准确，因为终端节点中多数类的分布相当高。下面对一个基本的分类器进行评价，以验证结果：

```
//使用10折交叉验证检测模型准确度
Evaluation eval = new Evaluation(data);
eval.crossValidateModel(model, data, 10, new Random(1), new
  String[] {});
System.out.println("Model performance:\n"+
  eval.toSummaryString());
```

输出如下模型性能：

```
Correctly Classified Instances         226               92.623  %
Incorrectly Classified Instances        18                7.377  %
Kappa statistic                          0.8839
Mean absolute error                      0.0421
Root mean squared error                  0.1897
Relative absolute error                 13.1828 %
Root relative squared error             47.519  %
Coverage of cases (0.95 level)          93.0328 %
Mean rel. region size (0.95 level)      27.8689 %
Total Number of Instances              244
```

从上述结果可以看到，模型做分类的准确度非常高，达到令人吃惊的92.62%。之所以能有这样好的结果，一个重要原因在于我们的评价设计。我的意思是：这些连续的实例彼此非常类似，如果在10折交叉验证中对其随机划分，很有可能会让训练与测试所用的实例几乎完全相同。因此，使用简单的*k*折交叉验证评估模型性能时，结果很乐观。

一个更好的方法是，使用对应于不同组测度或者不同人群的折。比如，可以使用应用程序收集5个人的学习数据，然后运行*k*人交叉验证（k-person cross validation）。模型使用4个人的学习数据做训练，然后借助第5个人做测试。针对每个人重复这一过程，并且对结果做平均。这样会让我们对模型性能的评估更加合理。

先将模型评估的讨论搁置，下面看看如何处理分类器错误。

### 9.3.1　减少假性转换

在行为识别流程的最后一步，我们要确保分类不会太易变。也就是说，我们不希望行为每毫秒都发生改变。对于这个问题，一个基本解决方法是设计一个过滤器，过滤行为序列中快速改变的行为。

创建一个过滤器，记住最后窗口活动，并返回最频繁的行为。如果有多个行为拥有相同分数，它会返回最近一个。

首先，新建一个类`SpuriousActivityRemoval`，它包含一个行为列表与`window`参数：

```
class SpuriousActivityRemoval{

  List<Object> last;
  int window;

  public SpuriousActivityRemoval(int window){
    this.last = new ArrayList<Object>();
    this.window = window;
  }
```

接着，创建`Object filter(Object)`方法，它接收一个行为，返回经过过滤的行为。这个方法先检查我们是否有足够的观测值，如果没有，它会简单地存储传入的观测值并将其返回，代码如下：

```
  public Object filter(Object obj){
    if(last.size() < window){
      last.add(obj);
      return obj;
  }
```

如果已经收集了`window`个观测值，那么简单返回最频繁的观测值，移除最旧的观测值，并插入新观测值。

```
    Object o = getMostFrequentElement(last);
    last.add(obj);
    last.remove(0);
    return o;
  }
```

上述代码用到了`getMostFrequentElement()`方法，它返回列表中最频繁的对象。可以使用HashMap实现这个方法，代码如下：

```
  private Object getMostFrequentElement(List<Object> list){

    HashMap<String, Integer> objectCounts = new HashMap<String,
      Integer>();
    Integer frequntCount = 0;
    Object frequentObject = null;
```

接下来，遍历链表中的所有元素，将每个唯一元素插入HashMap。如果它已经存在于HashMap中，则更新计数。在循环的最后，存储目前找到的最频繁的对象，代码如下：

```
    for(Object obj : list){
      String key = obj.toString();
      Integer count = objectCounts.get(key);
      if(count == null){
        count = 0;
      }
      objectCounts.put(key, ++count);

      if(count >= frequntCount){
        frequntCount = count;
        frequentObject = obj;
      }
    }

    return frequentObject;
  }

}
```

运行一个简单的例子：

```
String[] activities = new String[]{"Walk", "Walk", "Walk", "Run",
  "Walk", "Run", "Run", "Sit", "Sit", "Sit"};
SpuriousActivityRemoval dlpFilter = new
  SpuriousActivityRemoval(3);
for(String str : activities){
  System.out.println(str +" -> "+ dlpFilter.filter(str));
}
```

上面这个例子输出如下行为：

```
Walk -> Walk
Walk -> Walk
Walk -> Walk
```

9

```
Run -> Walk
Walk -> Walk
Run -> Walk
Run -> Run
Sit -> Run
Sit -> Run
Sit -> Sit
```

输出结果是一个连续的行为序列，也就是说，我们没做快速改变。这会带来一些延迟，但可以接受，除非它对应用程序至关重要。

将分类器识别过的*n*个过往行为追加到特征向量，由此可以增强行为识别的能力。但这样做会带来风险，即可能会让机器学习算法认为当前行为总是跟前一个行为一样，这种情况时常发生。对于这个问题，我们可以使用两个分类器（A与B）加以解决：分类器B的属性向量包含*n*个过往行为，这些过往行为由分类器A识别。分类器A的属性向量不包含任何过往行为。这种情形下，即使B为过往行为给出很大权重，分类器A识别的过往行为也会变，因为分类器A不受分类器B的影响。

接下来要做的就是，将分类器与过滤器嵌入我们的移动应用。

### 9.3.2　将分类器嵌入移动应用

有两种方法可以将一个分类器嵌入移动应用。第一种方法是，以Weka格式导出一个模型，将Weka库用作移动应用的一个依赖、加载模型等。整个过程和第3章看到的例子是一样的。第二个方法更加轻量化，将模型以源代码形式导出，比如创建一个类，实现决策树分类器。然后将源代码复制粘贴到移动应用，没有任何导入Weka依赖的事件。

幸运的是，使用`toSource(String)`函数可以很轻松地将一些Weka模型导出为源代码。

```
//输出实现了决策树的源代码
System.out.println("Source code:\n" +
  model.toSource("ActivityRecognitionEngine"));
```

上述代码将我们的模型输出为一个`ActivityRecognitionEngine`类。接下来，详细看看输出得到的ActivityRecognitionEngine类的代码：

```
class ActivityRecognitionEngine {

  public static double classify(Object[] i)
    throws Exception {

    double p = Double.NaN;
    p = ActivityRecognitionEngine.N17a7cec20(i);
    return p;
  }
  static double N17a7cec20(Object []i) {
```

```
        double p = Double.NaN;
        if (i[64] == null) {
          p = 1;
        } else if (((Double) i[64]).doubleValue() <= 10.353474) {
        p = ActivityRecognitionEngine.N65b3120a1(i);
        } else if (((Double) i[64]).doubleValue() > 10.353474) {
          p = 2;
        }
        return p;
      }
    ...
```

ActivityRecognitionEngine类实现了我们之前讨论的决策树。机器随生成的函数名（比如N17a7cec20(Object [])）对应于决策树的节点。这个分类器可以通过classify(Object[])方法进行调用，并且调用时要传入一个特征向量。这个特征向量可以通过我们前面介绍的过程获得，它返回一个double值，表示类标签索引。

## 9.4　小结

本章讨论了如何为移动应用实现一个行为识别模型。我们了解了整个过程，包括数据收集、特征提取、建立模型、评价模型以及部署模型。

下一章将学习另一个Java库——Mallet，它用于进行文本分析。

第 10 章

# 利用Mallet进行文本挖掘——主题模型与垃圾邮件检测

本章先讨论文本挖掘的定义以及可以进行的分析，以及为何将其应用于应用程序。再讨论如何使用Mallet——一个处理自然语言的Java库，包括数据导入与文本预处理。然后了解文本挖掘的两个具体应用：主题模型与垃圾邮件检测，在主题模型中，我们将讨论如何使用文本挖掘技术从不曾阅读过的文本文档中识别主题；在垃圾邮件检测中，我们将讨论如何自动为文本文档进行分类。

本章内容涵盖如下主题：

- 文本挖掘简介
- 安装与使用Mallet
- 主题模型
- 垃圾邮件检测

## 10.1　文本挖掘简介

文本挖掘也叫文本分析，指的是从文本文档自动提取高质量信息的过程。这些文本文档大部分是使用自然语言写成的，高质量信息是那些紧密相关、新颖又有趣的信息。

一个典型的文本分析应用是扫描一组文档产生搜索索引，文本挖掘可以应用到其他许多领域，包括文本分类（分类到特定域）、文本聚类（自动组织一组文档）、情绪分析（识别与提取文档中的主观信息）、概念/实体提取（可以从文档中识别人、地点、组织、其他实体）、文档摘要（自动给出源文档中最重要的观点）、学习命名实体间的关系。

基于统计模式挖掘的过程通常包含如下步骤：

(1) 信息检索与提取；
(2) 将无结构文本数据转换为有结构数据，比如分析、移除噪声单词、词汇分析、计算词频、

撷取语言特征等；

(3) 从结构化数据与标注/注释发现模式；

(4) 评价与解释结果。

本章后半部分将学习文本挖掘的两个应用：主题模型与文本分类。接下来，先看看它们能做什么。

### 10.1.1 主题模型

主题模型是一种无监督技术，如果你需要分析一个包含大量文本文档的档案，希望了解该档案包含的内容，但又不想亲自阅读每个文档，这时主题模型就很有用。文本文档可以是博客文章、电子邮件、推文、文件、图书章节、日记等。主题模型在文本语料库中寻找模式，更准确地说，它采用一种有统计意义的方式识别主题，并组成单词表。最有名的算法是**隐含狄利克雷分布**（**Latent Dirichlet Allocation**，Blei等，2003），它假设作者从可能的单词篮中选择单词并组成一段文字，每个篮子对应一个主题。借助这个假设，这个算法可以从数学上把文本拆解到相应篮子（baskets），这些篮子是拆解后的单词最有可能的来源。然后算法不断迭代这个过程，直到它将所有词分配到最有可能的篮子，我们把这些篮子叫作“主题”。

比如，如果针对一系列新闻文章使用主题模型，算法将会返回一系列主题以及最有可能组成这些主题的关键字。借用新闻文章的例子，列表看起来可能像下面这样：

- Winner（获胜者）、goal（射门）、football（足球）、score（得分）、first place（第一名）
- Company（公司）、stocks（股票）、bank（银行）、credit（信用）、business（生意）
- Election（选举）、opponent（对手）、president（总统）、debate（辩论）upcoming（即将来临）

通过浏览关键字，我们能够知道这些新闻文章与体育、商业、即将到来的选举有关。本章后半部分将通过这个新闻文章的例子学习实现主题模型的方法。

### 10.1.2 文本分类

文本分类的目标是根据文本文档的内容，将一个文本文档划入一个或多个分类。这些分类往往是一个更普通的主题，比如车辆、宠物。这些普通的类别就是主题，分类任务也叫文本分类、主题分类、主题发现。虽然我们可以根据文档类型、作者、印刷年份等属性对文档进行分类，但本章学习的重点是只根据文档内容进行分类。文本分类的例子如下。



- 对电子邮件、用户评论、网页等垃圾内容的检测
- 色情内容检测

- ❑ 情绪检测：自动将用户对一个产品或服务的评论划分为正面评论与负面评论
- ❑ 根据电子邮件内容对电子邮件进行分类
- ❑ 专题搜索：搜索引擎将搜索限制到某个特定主题或类型以提供更准确的结果

这些例子表明文本分类在信息检索系统中非常重要，因此大部分现代信息检索系统都在使用某种类型的文本分类器。本书所用的分类任务的例子是通过文本分类检测垃圾邮件。

本章还会介绍Mallet，它是一个Java包，用于执行自然语言统计处理、文档分类、聚类、主题模型、信息提取，以及其他针对文本的机器学习应用。然后学习文本分析（文本分类）的两个应用：主题模型与垃圾邮件检测。

## 10.2　安装 Mallet

进入UMass Amherst University网站（http://mallet.cs.umass.edu/download.php）下载Mallet。转到**Download**页面，选择最新稳定版本（写作本书时是2.0.8）下载即可，如图10-1所示。

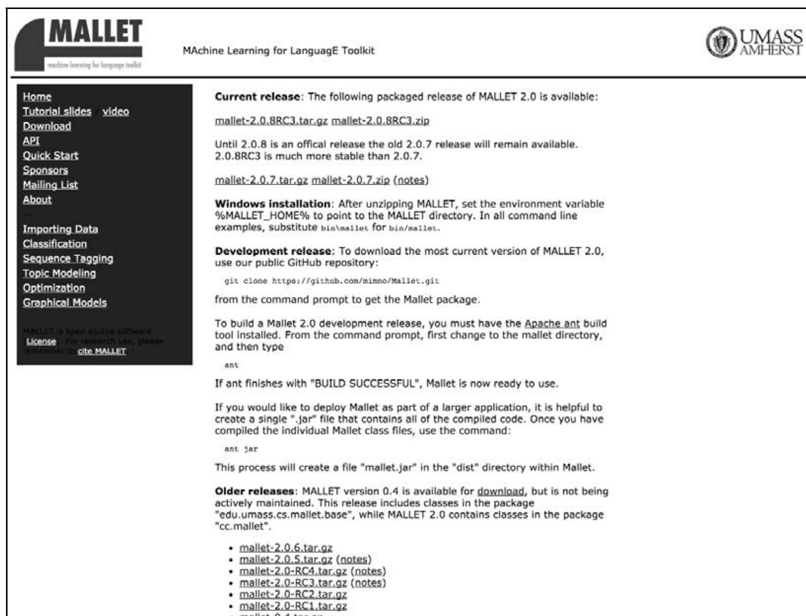



图10-1　下载Mallet

下载ZIP文件后，进行解压缩。在解压后的目录中，你应该能够看到一个名为`dist`的文件夹，里面有两个JAR文件：`mallet.jar`与`mallet-deps.jar`。其中，mallet.jar文件包含所有打包好的Mallet类，mallet-deps.jar包含所有依赖文件。请将这两个JAR文件放入你的项目，当作引用包，如图10-2所示。

| Name | Size | Kind |
|---|---|---|
| bin | -- | Folder |
| build.xml | 3 KB | XML |
| class | -- | Folder |
| dist | -- | Folder |
| mallet-deps.jar | 2,6 MB | Java JAR file |
| mallet.jar | 2,2 MB | Java JAR file |
| lib | -- | Folder |
| LICENSE | 12 KB | TextEd...ument |
| Makefile | 4 KB | TextEd...ument |
| pom.xml | 3 KB | XML |
| README.md | 2 KB | Markd...cument |
| sample-data | -- | Folder |
| src | -- | Folder |
| stoplists | -- | Folder |
| test | -- | Folder |

图10-2　将JAR文件放入项目

如果你使用的是Eclipse，那么在**Project**上点击右键，选择**Properties**，再选择**Java Build Path**。在**Libraries**选项卡中，点击**Add External JARs**，然后选择上面两个JAR文件，并进行确认，如图10-3所示。

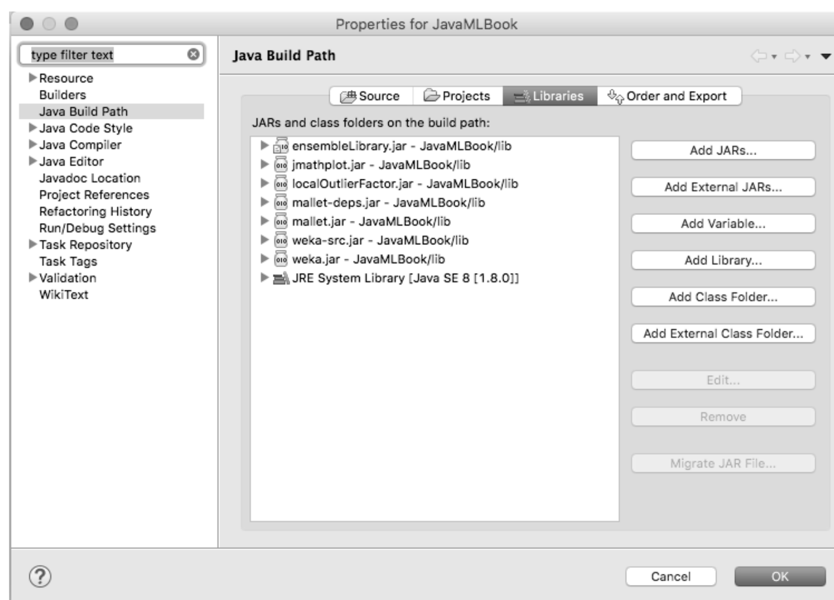



图10-3　选择并确认JAR文件

接下来，准备开始使用Mallet。



## 10.3　使用文本数据

文本挖掘的主要挑战之一是，将没有结构的自然语言转换为结构化的基于属性的实例。这个

过程包含许多步骤，如图10-4所示。

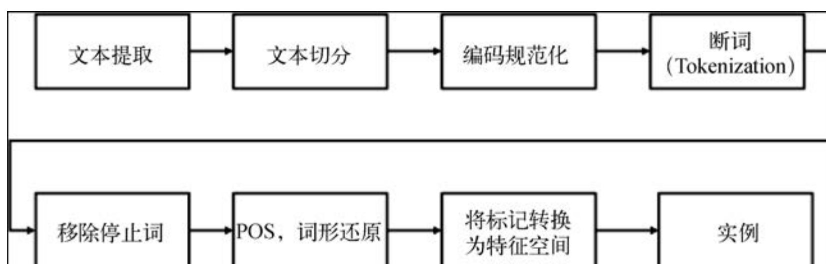



图10-4　文本挖掘步骤

首先，从网络、现有文档或数据库提取一些文本。在第一步的最后，文本仍然是XML格式或其他某些专用格式。因此，接下来的一步是提取实际文本，并将其划分到文档的各个部分，比如题目、大标题、摘要、正文等。第三步是规范文本编码，保证字符用相同方式表示，比如将ASCII、ISO 8859-1、Windows-1250编码格式的文档转换为Unicode编码。接着，通过断词将文档分割为特定词，然后移除那些具有较低预测能力的常用词，比如the、a、I、We等。

**词性**（**POS**）标注与词形还原通过移除词尾和修饰符，将每个标记（即单词）转换为其基本形式，即词根，比如将running变化为run、将better变为good等。一个简化的方法是词干提取，它针对的是单个词。关于如何使用这个特定的词，没有任何上下文，因此它不能区分带有不同含义的词，而主要依赖于词性，比如axes既是axe的复数，也是axis的复数。

最后一步是将标记转换为特征空间。通常，特征空间是一个**词袋**（**BoW**）表示。这个表示中，出现在数据集中的所有单词组都会被创建，即词袋。每个文档表示为一个向量，记录某个特定单词在文档中出现的次数。

请看下面两个句子：

- Jacob likes table tennis. Emma likes table tennis too.（雅各布喜欢乒乓球。艾玛也喜欢乒乓球）
- Jacob also likes basketball.（雅各布也喜欢篮球。）

这个例子中，词袋为`{ Jacob, likes, table, tennis, Emma, too, also, basketball }`，包含8个不同单词。现在，可以使用列表索引将这两个句子表示为向量，表示文档中的一个单词在特定索引位置出现的次数，如下：

- `[1, 2, 2, 2, 1, 0, 0, 0]`
- `[1, 1, 0, 0, 0, 0, 1, 1]`

最后，将这样的向量变成实例，以做进一步学习。



另一种基于BoW模型的表示方法——word2vec也非常强大。Word2vec由Google公司的Tomas Mikolov领导的研究小组在2013年提出。Word2vec是一个神经网络，用来学习单词的词向量（distributed representations）。这种表示法的一种有趣特性是，单词存在于簇中，这样一些单词关系（比如类比）就可以使用向量数学再现。一个著名的例子是king–man+woman= queen。

更多细节与实现请前往如下网址学习：

https://code.google.com/archive/p/word2vec/

### 10.3.1 导入数据

本章不会学习如何从网站抓取一组文档或者从数据库提取它们，此处假设已经收集好了这组文档，并将其保存在`.txt`文件中。接下来，了解一下加载它们时的两种情况：第一种情况是每个文档存储在各自的`.txt`文件中；另一种情况是所有文档都存在一个文件中，每行就是一个文档。

#### 1. 从目录导入

Mallet提供了`cc.mallet.pipe.iterator. FileIterator`类，支持从路径读取文件。文件迭代器带有如下3个参数：

- 包含文本文件的`File[]`目录列表
- 文件过滤器，用于指定选择目录中的哪些文件
- 要应用到文件名的模式，用于产生一个类标签

图10-5显示了文件夹中的数据文件，这些文档按文件夹组织成5个主题（`tech`、`entertainment`、`politics`、`sport`、`business`）。每个文件夹包含与特定主题相关的文档。

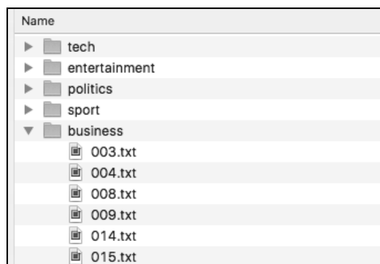



图10-5 数据文件

此情形下，初始化`iterator`，代码如下：

```
FileIterator iterator =
  new FileIterator(new File[]{new File("path-to-my-dataset")},
```

```
  new TxtFilter(),
  FileIterator.LAST_DIRECTORY);
```

第一个参数指定根文件夹的路径，第二个参数将迭代器限制在.txt文件上，最后一个参数让方法将路径中的最后目录名用作类标签。

### 2. 从文件导入数据

加载文档的另外一种方法是使用cc.mallet.pipe.iterator.CsvIterator.CsvIterator (Reader, Pattern, int, int, int)，它假定所有文档位于一个文件，每行返回一个实例，通过一个正则表达式进行提取。初始化这个类时，需要提供如下参数。

- Reader：这个对象指定如何从文件读取数据。
- Pattern：这是一个正则表达式，提取3个组：数据、目标标签、文档名。
- int, int, int：这些是数据、目标、名称组的索引，它们出现在上面的正则表达式中。

假设有一个文本文档，它有指定的文档名、分类、内容，格式如下：

```
AP881218 local-news A 16-year-old student at a private
  Baptist...
AP880224 business The Bechtel Group Inc. offered in 1985 to...
AP881017 local-news A gunman took a 74-year-old woman hostage...
AP900117 entertainment Cupid has a new message for lovers
  this...
AP880405 politics The Reagan administration is weighing w...
```

使用如下正则表达式，将文档的每一行解析成3个组：

```
^(\\S*)[\\s,]*(\\S*)[\\s,]*(.*)$
```

有3个组出现在圆括号()中，其中第三组包含数据，第二组包含目标类别，第一组包含文档ID。迭代器初始化如下：

```
CsvIterator iterator = new CsvIterator (
fileReader,
Pattern.compile("^(\\S*)[\\s,]*(\\S*)[\\s,]*(.*)$"),
  3, 2, 1));
```

上面代码中，正则表达式用于提取3个组，采用空格分隔，它们的顺序为3、2、1。

接下来进入数据预处理流程。

## 10.3.2 对文本数据做预处理

对遍历数据的迭代器做好初始化后，需要对数据做一系列变换，这在开头部分已经提到过。对此，Mallet提供了相应的处理流程，其中包含各种步骤，在cc.mallet.pipe包中可以找到它们。下面给出了一些例子。

- ❑ Input2CharSequence：从各种文本源（URI、File、Reader）读取数据，并转换为CharSequence。
- ❑ CharSequenceRemoveHTML：从CharSequence移走HTML。
- ❑ MakeAmpersandXMLFriendly：将符号序列的标记中的&转换为&amp。
- ❑ TokenSequenceLowercase：将数据域符号序列中每个标记的文本转换成小写。
- ❑ TokenSequence2FeatureSequence：将每个实例数据域中的符号序列转换为特征序列。
- ❑ TokenSequenceNGrams：将数据域中的符号序列转换为带标记的ngrams序列，即两个或更多个单词组合。



关于处理步骤的完整列表，可以在下面Mallet文档中看到：
http://mallet.cs.umass.edu/api/index.html?cc/mallet/pipe/iterator/package-tree.html

接下来，创建用于导入数据的类。

首先创建一个管道（pipeline），每个处理步骤对应于Mallet中的一个管道。可以用串行方式把管道连接起来，形成ArrayList<Pipe>对象列表。

```
ArrayList<Pipe> pipeList = new ArrayList<Pipe>();
```

先从一个文件对象读取数据，将所有字符转换为小写：

```
pipeList.add(new Input2CharSequence("UTF-8"));
pipeList.add( new CharSequenceLowercase() );
```

接下来，使用正则表达式对原始字符串进行标记化。下面模式包括Unicode字母、数字以及下划线字符：

```
Pattern tokenPattern =
  Pattern.compile("[\\p{L}\\p{N}_]+");

pipeList.add(new CharSequence2TokenSequence(tokenPattern));
```

使用标准的英文停止词表，移走停止词，即那些不具预测能力的高频词。其他两个参数分别指定移走停止词时是否区分大小写，以及删除单词后是否标记。将这两个参数全部设置为false：

```
pipeList.add(new TokenSequenceRemoveStopwords(false, false));
```

我们不会保存实际单词，而将它们变成整数，表示单词在词袋中的索引。

```
pipeList.add(new TokenSequence2FeatureSequence());
```

对于类标签做同样处理，即不用标签字符串而使用一个整数，指示标签在词袋中的位置。



```
pipeList.add(new Target2Label());
```

可以通过调用PrintInputAndTarget打印特征与标签。

```
pipeList.add(new PrintInputAndTarget());
```

最后，将管道列表存储到SerialPipes类，这个类通过一系列管道转换实例。

```
SerialPipes pipeline = new SerialPipes(pipeList);
```

接下来，看看如何将其应用到文本挖掘应用。

## 10.4　为 BBC 新闻做主题模型

如前所述，主题模型的目标是识别文本语料库（对应于文档主题）中的模式。这个例子中，我们使用的数据集来自BBC新闻。这个数据集是机器学习研究中常用的基准测试数据集之一，仅用于非商业与研究目的。

我们的目标是创建一个分类器，用它为未分类的文档指派一个标题。

### 10.4.1　BBC 数据集

为了研究基于支持向量机的文档聚类问题，Greene与Cunningham（2006）采集了BBC新闻数据并制成BBC数据集。这个数据集包含2225个文档，它们全部来自BBC新闻网站，时间跨度为2004~2005年，可划分为5个主题：商业、环境、政治、体育、技术。可以从如下网站获得这个数据集：

http://mlg.ucd.ie/datasets/bbc.html

从**Dataset: BBC**部分下载原始文本文件。此外，你可能已经注意到，这个网站还包含已经做过预处理的数据集，但我们不会下载它，因为我们想自己动手处理数据集。下载后的ZIP文件中包含5个文件夹，每个文件夹对应一个主题，实际文档位于相应的主题文件夹之下，如图10-6所示。

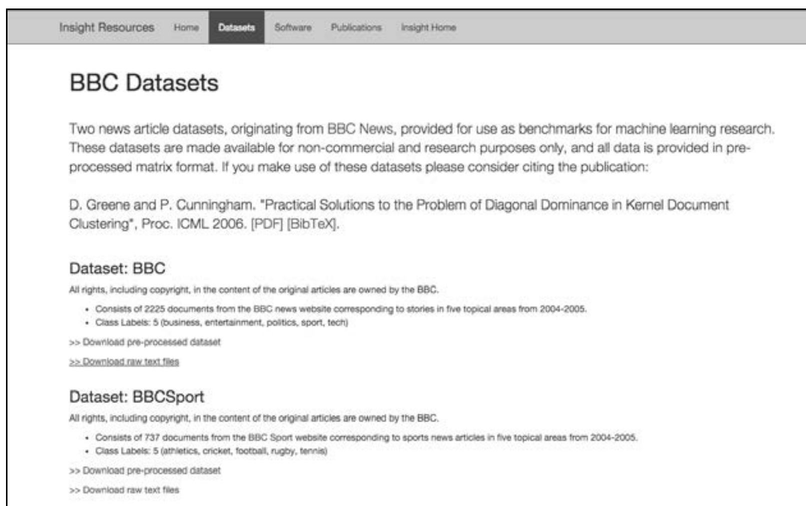



图10-6　原始文本文件

下面创建主题分类器。

### 10.4.2　建模

首先，导入数据集，并对文本做处理：

```
import cc.mallet.types.*;
import cc.mallet.pipe.*;
import cc.mallet.pipe.iterator.*;
import cc.mallet.topics.*;

import java.util.*;
import java.util.regex.*;
import java.io.*;

public class TopicModeling {

  public static void main(String[] args) throws Exception {

String dataFolderPath = args[0];
String stopListFilePath = args[1];
```

然后，创建一个默认管道（如前所述）：

```
ArrayList<Pipe> pipeList = new ArrayList<Pipe>();
pipeList.add(new Input2CharSequence("UTF-8"));
Pattern tokenPattern = Pattern.compile("[\\p{L}\\p{N}_]+");
pipeList.add(new CharSequence2TokenSequence(tokenPattern));
pipeList.add(new TokenSequenceLowercase());
pipeList.add(new TokenSequenceRemoveStopwords(new
  File(stopListFilePath), "utf-8", false, false, false));
pipeList.add(new TokenSequence2FeatureSequence());
pipeList.add(new Target2Label());
SerialPipes pipeline = new SerialPipes(pipeList);
```

接着，初始化`folderIterator`：

```
FileIterator folderIterator = new FileIterator(
    new File[] {new File(dataFolderPath)},
    new TxtFilter(),
    FileIterator.LAST_DIRECTORY);
```

新建实例列表，将我们想用于处理文本的管道传递给它：

```
InstanceList instances = new InstanceList(pipeline);
```

最后，处理迭代器给出的每个实例：

```
instances.addThruPipe(folderIterator);
```

下面使用`cc.mallet.topics.ParallelTopicModel.ParallelTopicModel`类（实现了

一个简单的**Latent Dirichlet Allocation**模型）创建带有5个主题的模型。LDA是主题模型的通用方法，它使用狄利克雷分布评估选定主题产生特定文档的可能性。本章不会涉及相关细节，感兴趣的读者请参考D. Blei等人发表的原论文（2003）。请注意，LDA这个缩写还有可能指机器学习中另外一种分类算法——**线性判别分析**（**Linear Discriminant Analysis**），但它们只是缩写一样，此外再无共同之处。

对`ParallelTopicModel`类进行实例化时，有alpha与beta参数，基本含义如下。

- 高alpha值表示每个文档可能混合多个主题，不特指某一个主题；低alpha值使文档较少受这些条件的约束，这意味着文档可能混合几个主题，也可能只有一个主题。
- 高beta值表示每个主题可能混合很多单词，不是特定某个单词；低beta值表示主题只混合几个单词。

例子中，我们将这两个参数设置得小一些（alpha_t = 0.01，beta_w = 0.01）。因为我们假设数据集中的主题混合得不多，并且每个主题有很多单词：

```
int numTopics = 5;
ParallelTopicModel model =
new ParallelTopicModel(numTopics, 0.01, 0.01);
```

接下来，将实例添加到模型。由于我们使用的是并行实现，所以还要指定并行执行的线程数，代码如下：

```
model.addInstances(instances);
model.setNumThreads(4);
```

按照选定的迭代次数运行模型。每次迭代都是为了更好地评估内部LDA参数。测试时，我们可以指定较少的迭代次数，比如50次；而实际应用中，通常将迭代测试设置为`1000`或`2000`次。最后，调用`void estimate()`方法，实际创建一个LDA模型：

```
model.setNumIterations(1000);
model.estimate();
```

模型输出结果如下：

```
0 0,06654  game england year time win world 6
1 0,0863  year 1 company market growth economy firm
2 0,05981  people technology mobile mr games users music
3 0,05744  film year music show awards award won
4 0,11395  mr government people labour election party blair

[beta: 0,11328]
<1000> LL/token: -8,63377

Total time: 45 seconds
```

`LL/token`指模型的对数相似度（log-likelihood）除以标记总数，表示数据与给定模型的相似

程度，该值越大，表示模型品质越高。

输出也显示了描述每个主题的热门词汇，这些词汇与初始主题有很好的对应。

- **Topic 0**: game, England, year, time, win, world, 6 → sport
- **Topic 1**: year, 1, company, market, growth, economy, firm → finance
- **Topic 2**: people, technology, mobile, mr, games, users, music → tech
- **Topic 3**: film, year, music, show, awards, award, won → entertainment
- **Topic 4**: mr, government, people, labor, election, party, blair → politics

其中，有些词汇意义不大，比如mr、1、6，我们可以将其放入停止词列表。此外，有些词出现了两次，比如award与awards。之所以会出现这种情况是因为，我们没有使用任何词干分析器与词形还原管道。

接下来，对模型的性能进行评估。

### 10.4.3　评估模型

统计上的主题模型拥有非监督特征，这使选择模型变得困难。对于某些应用，可能有一些非本质的任务要处理，比如信息检索或文档分类，我们可以为其评估性能。但我们通常希望评估的是模型概括主题的能力，而不是这些任务。

Wallach等人（2009）提出了一种衡量模型质量的方法，这种方法计算的是模型下抽取文档的对数概率，将未见过的文档的可能性用来比较模型——可能性越高，表示模型越好。

首先，将文档分成训练集与测试集（即留存文档），把90%的文档用来训练，10%的文档用来测试。

```
// 划分数据集
InstanceList[] instanceSplit= instances.split(new Randoms(),
  new    double[] {0.9, 0.1, 0.0});
```

接着，使用90%的文档重建模型：

```
// 使用前90%做训练
model.addInstances(instanceSplit[0]);
model.setNumThreads(4);
model.setNumIterations(50);
model.estimate();
```

然后，初始化评价器`MarginalProbEstimator`，它实现了留存文档的Wallach对数概率：

```
// 获取评价器
MarginalProbEstimator estimator = model.getProbEstimator();
```



Annalyn Ng在她的博客中，对LDA做了直观说明：https://annalyzin.wordpress.com/2015/06/21/laymans-explanation-of-topic-modeling-withlda-2/

为了深入了解LDA算法以及组件与工作原理，可以阅读David Blei等人撰写的LDA论文：http://jmlr.csail.mit.edu/papers/v3/blei03a.html。或者阅读布朗大学D. Santhanam所做的总结演示：http://www.cs.brown.edu/courses/csci2950-p/spring2010/lectures/2010-03-03_santhanam.pdf。

这个类实现了许多评价器，需要非常深厚的理论知识才能理解LDA方法的工作原理。我们从中选择从左到右（left-to-right）的评价器，它的适用范围很广，比如文本挖掘、语音识别等。从左到右的评价器实现为`double evaluateLeftToRight`方法，接收如下4个参数。

- `Instances heldOutDocuments`：测试实例。
- `int numParticles`：这个算法参数指从左到右的标记数量，默认值是`10`。
- `boolean useResampling`：表示在从左到右的评价中是否对主题重采样。重采样会提高准确度，但会导致文档长度呈指数增长。
- `PrintStream docProbabilityStream`：这是个文件或`stdout`，我们将为每个文档推算的对数概率写入其中。

运行评价器，代码如下：

```
double loglike = estimator.evaluateLeftToRight(
  instanceSplit[1], 10, false, null););
System.out.println("Total log likelihood: "+loglike);
```

我们的例子中，评价器输出如下对数似然值。与使用其他参数、流水线或数据创建的模型进行比较时，这个值才有意义。对数似然值越高，模型越好：

```
Total time: 3 seconds
Topic Evaluator: 5 topics, 3 topic bits, 111 topic mask
Total log likelihood: -360849.4240795393
Total log likelihood
```

接下来，看看如何使用这个模型。

## 10.4.4　重用模型

我们通常不会在运行中创建模型，更常见的做法是，训练一次模型，然后重复用它对新数据做分类。

请注意，如果你打算对新文档做分类，它们也需要像其他文档一样通过一样的流水线——对于训练与分类，管道需要是一样的。训练期间，管道数据字母表随每个训练实例更新。如果你使

用相同步骤创建了一个新管道，那么不会得到相同的流水线，因为它的数据字母表是空的。因此，为了将模型应用到新数据，要随模型一起保存/加载管道，并且使用这个管道添加新实例。

### 1. 保存模型

Mallet提供了一个标准方法，它基于序列化保存与恢复对象。新建一个ObjectOutputStream类的实例，然后将实例对象写入文件，代码如下：

```
String modelPath = "myTopicModel";

// 保存模型
ObjectOutputStream oos = new ObjectOutputStream(
new FileOutputStream (new File(modelPath+".model")));
oos.writeObject(model);
oos.close();

// 保存流水线
oos = new ObjectOutputStream(
new FileOutputStream (new File(modelPath+".pipeline")));
oos.writeObject(pipeline);
oos.close();
```

### 2. 恢复模型

相对于通过序列化保存模型，使用ObjectInputStream类恢复模型是一个相反操作：

```
String modelPath = "myTopicModel";

// 加载模型
ObjectInputStream ois = new ObjectInputStream(
  new FileInputStream (new File(modelPath+".model")));
ParallelTopicModel model = (ParallelTopicModel) ois.readObject();
ois.close();

// 加载流水线
ois = new ObjectInputStream(
  new FileInputStream (new File(modelPath+".pipeline")));
SerialPipes pipeline = (SerialPipes) ois.readObject();
ois.close();
```

上面讨论了如何创建一个LDA模型，并根据各主题自动对文档进行分类。接下来的例子中，我们将学习另外一个文本挖掘问题——文本分类。

## 10.5 垃圾邮件检测



垃圾信息主要指那些未经用户同意发送来的信息，这些信息包含广告内容、受病毒感染的附件以及跳转到钓鱼网站或恶意网站的链接等。最常见的垃圾信息是垃圾邮件，除此之外，垃圾信息还出现在其他各种地方，比如网站评论区、即时信息、网络论坛、博客、在线广告等。

接下来，我们将讨论如何创建朴素贝叶斯垃圾邮件过滤器，使用词袋描述识别垃圾邮件。朴素贝叶斯垃圾邮件过滤器是基本技术之一，它在第一批商业垃圾邮件过滤器中被实现，比如**Mozilla Thunderbird**邮件客户端就使用了这样的过滤器实现。虽然我们列举的是过滤垃圾邮件的例子，但这些方法也可以用于过滤其他类型的文本垃圾。

### 10.5.1　垃圾邮件数据集

Androutsopoulos等人（2000）收集并制作了第一批垃圾邮件数据集之一，他们使用它测试垃圾邮件过滤算法。这些人研究如何使用朴素贝叶斯分类器检测垃圾邮件，并且研究使用其他管道（比如停用词表、词干分析器、词形还原）是否有利于提高过滤器的性能。Andrew Ng在OpenClassroom的机器学习课堂上对数据集进行重新组织，你可以从如下页面下载：http://openclassroom.stanford.edu/MainFolder/DocumentPage.php?course=MachineLearning&doc=exercises/ex6/ex6.html。

选择并下载第二个——`ex6DataEmails.zip`，如图10-7所示。

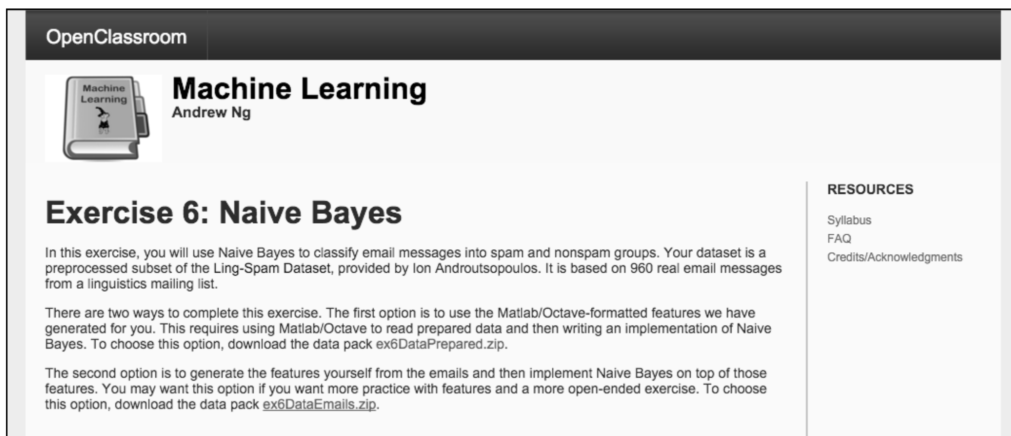



图10-7　选择并下载ex6DataEmails.zip文件

ZIP包含如下4个文件夹（Ng，2015）。

- `nonspam-train`与`spam-train`文件夹包含经过预处理的电子邮件，它们用于训练。每个文件夹包含350封电子邮件。
- `nonspam-test`与`spam-test`文件夹是测试集，包含130封垃圾邮件与130封非垃圾邮件。你将针对这些文档进行预测。请注意，尽管有单独的文件夹告诉你正确的标签，但对这些测试文档做预测时请不要使用它们。做完预测后，你可以使用正确的标签检查分类是否正确。

为了利用Mallet的文件夹迭代器，需要重新组织文件夹结构，如下所示。创建两个文件夹——train与test，将spam/nospam文件夹放入相应文件夹。文件夹最初结构如图10-8所示。

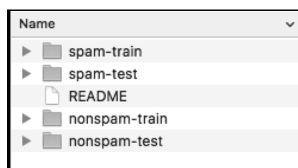



图10-8　文件夹最初结构

经过调整，文件夹的结构如图10-9所示。

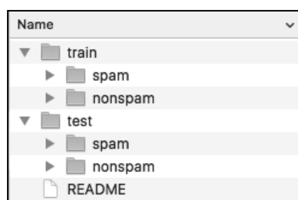



图10-9　调整后的文件夹结构

接下来，将e-mail信息转换为特征向量。

## 10.5.2　特征生成

根据前面的讲解，创建一个默认流水线（pipeline）。

```
ArrayList<Pipe> pipeList = new ArrayList<Pipe>();
pipeList.add(new Input2CharSequence("UTF-8"));
Pattern tokenPattern = Pattern.compile("[\\p{L}\\p{N}_]+");
pipeList.add(new CharSequence2TokenSequence(tokenPattern));
pipeList.add(new TokenSequenceLowercase());
pipeList.add(new TokenSequenceRemoveStopwords(new
  File(stopListFilePath), "utf-8", false, false, false));
pipeList.add(new TokenSequence2FeatureSequence());
pipeList.add(new FeatureSequence2FeatureVector());
pipeList.add(new Target2Label());
SerialPipes pipeline = new SerialPipes(pipeList);
```

请注意，我们额外添加了一个FeatureSequence2FeatureVector管道，将特征序列转换为特征向量。特征向量中有数据时，可以使用前面学习的任何一种分类算法。接下来继续我们的例子，演示如何在Mallet中创建一个分类模型。

接着，初始化一个文件夹迭代器，加载train文件夹中的样例。train文件夹包含spam与nonspam两个子文件夹，里面包含电子邮件样例，文件夹用作样例标签：

```
FileIterator folderIterator = new FileIterator(
    new File[] {new File(dataFolderPath)},
    new TxtFilter(),
    FileIterator.LAST_DIRECTORY);
```

使用流水线新建实例列表，用于处理文本：

```
InstanceList instances = new InstanceList(pipeline);
```

最后，处理迭代器提供的每一个实例：

```
instances.addThruPipe(folderIterator);
```

现在已经加载好数据，并且转换为特征向量。接下来，使用训练集训练模型，并在测试集`test`上做预测分类——`spam/nonspam`。

### 10.5.3　训练与测试模型

Mallet实现了一组分类器，它们位于`cc.mallet.classify`包，包括决策树、朴素贝叶斯、AdaBoost、bagging、boosting等。这里只讲一个基本的分类器——朴素贝叶斯分类器。先通过`ClassifierTrainer`类初始化分类器，再调用它的`train(Instances)`方法，即会返回分类器。

```
ClassifierTrainer classifierTrainer = new NaiveBayesTrainer();
Classifier classifier = classifierTrainer.train(instances);
```

接下来，了解一下这个分类器是如何工作的，以及如何在单独的数据集上评价其性能。

#### 模型性能

在单独的数据集上评价分类器之前，先导入`test`文件夹中的电子邮件：

```
InstanceList testInstances = new
  InstanceList(classifier.getInstancePipe());
folderIterator = new FileIterator(
    new File[] {new File(testFolderPath)},
    new TxtFilter(),
    FileIterator.LAST_DIRECTORY);
```

通过训练期间初始化的流水线传递数据：

```
testInstances.addThruPipe(folderIterator);
```

为了评价分类器性能，我们将用到`cc.mallet.classify.Trial`类，并且使用分类器与一系列测试样例对其进行初始化：

```
Trial trial = new Trial(classifier, testInstances);
```

初始化后立即执行评估。最后，可以只打印自己关心的指标。我们的例子中，我们想了解垃圾邮件信息分类的准确率与召回率，或者*F*值——返回两个值的调和平均数，代码如下：

```
System.out.println(
  "F1 for class 'spam': " + trial.getF1("spam"));
System.out.println(
  "Precision:" + trial.getPrecision(1));
System.out.println(
  "Recall:" + trial.getRecall(1));
```

评估结果输出如下：

```
F1 for class 'spam': 0.9731800766283524
Precision: 0.9694656488549618
Recall: 0.9769230769230769
```

从上述结果可以看到，模型发现垃圾信息的准确率是97.69%（recall）；标记为垃圾邮件的电子邮件中，准确率是96.94%。换言之，每100封垃圾邮件中，大约错判2个；每100封合法邮件中，有3封被错判为垃圾邮件。虽然结果并不十分完美，但却是一个很好的开始！

## 10.6　小结

本章讨论了文本挖掘与基于属性的传统学习方法的不同之处，需要大量预处理步骤，将书面的自然语言转换为特征向量。而且讨论了如何使用Mallet——一个基于Java的自然语言处理库——解决两个实际问题。首先，使用LDA模型为新闻语料库中的主题创建模型，通过它可以为新文档指派一个主题。此外，我们还讨论了如何使用词袋表示创建一个朴素贝叶斯垃圾邮件过滤器。

本章最后部分对如何应用各种类库解决机器学习任务做了技术演示。此处无法讲解更多有趣的应用，也不能在许多方面给出更多细节。关于如何继续学习以及深入了解相关主题，下一章给出了更多建议，供各位参考。

# 第 11 章 机器学习进阶

前面讲解了Java机器学习库以及如何使用它们解决实际问题，本章是整个学习旅程的最后一段。但无论如何，这不应该是你学习的终点。对于如何在真实世界中部署模型、会遇到什么困难，以及去哪里深化知识，本章将给你提供一些切实可行的建议。此外，本章还给出了一些参考站点与页面，从中你可以找到更多资源、资料、会议信息，以及更多与机器学习相关的技术。

本章内容涵盖如下主题：

- 机器学习在现实生活中的重要方面
- 标准与标记语言
- 云端机器学习
- Web资源与竞赛

## 11.1　现实生活中的机器学习

关于如何在实际生产环境中部署与维护模型，论文、学术报告、讲座通常不会提及。对于这些问题，本节将讲述需要考虑哪些方面。

### 11.1.1　噪声数据

事实上，数据通常包含一些错误与缺陷，它们由各种原因引起，比如测量误差、人为错误，以及对训练样本分类时产生的误判等。我们将数据中的这些错误与缺陷称为噪声。噪声也有可能产生于对缺失值进行处理的过程，比如使用一组加权实例（与缺失值的概率分布相对应）替换带有未知属性值的实例。学习数据包含的噪声数据最典型的影响是，降低模型对新数据的预测准确度，并且生成的复杂模型也很难被用户理解与解释。

### 11.1.2　类不平衡

我们已经在第7章遇到过类不平衡问题，主要讲解的内容是检测欺诈性保险索赔。当时面临

的困难是，数据集中有超过90%的数据描述的是正常行为活动，而欺诈实例仅占数据集的很小一部分。这种情况下，如果模型的预测结果总是正常，那么90%的情况下都是正确的。这个问题在实际情况中极其常见，并且可以在各种应用中观察到，比如欺诈检测、异常检测、医疗诊断、石油泄漏检测、面部识别等。

现在，我们已经知道了分类不平衡问题是什么以及成因，接下来看看应该如何解决。第一个解决方法是把精力放在测度而不是分类精确度上，比如召回率、准确率、*F*值。这些测度主要关注模型对少数类的预测准确度（召回率）以及误报比例（准确率）。另一个方法基于重采样，其主要思想是减少重复出现的实例数量，采用的方式是让新集合包含的类保持平衡比例。

### 11.1.3　特征选择困难

可以说，特征选择是建模中最困难的部分，这需要建模者拥有相关的领域知识以及对问题的深刻认识。然而，良好的特征具有如下属性。

- **可复用性**：良好特征应该可以在不同模型、应用程序与小组中重用。
- **可变换性**：我们应该能够使用一个操作（比如`log()`、`max()`）对特征进行变换，或者通过自定义计算把多个特征组合在一起。
- **可靠性**：特征应该很容易监测，并且有合适的单元测试能够最大限度减少bug和问题。
- **可解释性**：为了做前面这些行为，需要能够理解特征的含义以及解释它们的值。

采集的特征越好，结果就会越准确。

### 11.1.4　模型链

一些模型可能产生一个输出，这个输出在另一个模型中被用作特征。而且，可以使用多个模型——集成方法，将任一模型转换为特征。这是一种非常好的方法，它能让我们得到更好的结果，但同时也可能带来一些问题。你必须小心，确保你的模型输出可随时接受依赖。并且，要尽量避免反馈回路，因为它们可能在流水线中产生依赖与瓶颈。

### 11.1.5　评价的重要性

另一个重要方面是模型的评价。除非你想把模型应用于新数据，衡量业务目标，否则不需要做预测分析。评价技术（比如交叉验证与分离的训练/测试集）可以简单地划分测试数据，这只能让你估计模型如何执行。生活往往不会给你一个包含所有已定义情况的训练数据集，因此在真实数据集中定义这两个集合时，需要认真思考。

一天结束时，我们想提高业务目标，比如提高广告转化率、增加顾客对推荐商品的点击等。

为了衡量这种改进，执行A/B测试，针对拥有相同统计意义的族群，测量各种指标的不同，每种指标对应不同算法。有关产品的决策总是数据驱动的。



A/B测试方法用于对设计的两个版本做随机化访问实验：A对应于原始版本，控制实验；B对应于另一个版本。这个方法可以判断新版本的实际效果能否超过原版本。A/B测试应用广泛，范围涉及网站改版、销售邮件、广告搜索等。

Udacity有一个免费课程，讲解设计与A/B测试分析内容，可以在如下地址观看：https://www.udacity.com/course/abtesting--ud257。

### 11.1.6　从模型到产品

从在实验室中创建一个准确模型，到将其变为一个产品，整个过程需要紧密结合数据科学与工程技术，如下3个步骤与图11-1所示。

(1) **数据研究与构建假设**表示对问题建模，并且对模型做初步评价。

(2) **构建解决方案与实现**表示通过编写更高效、更稳定、更具扩展性的代码，让模型进入产品化流程。

(3) **在线评价**是最后一步，在业务目标下，使用A/B测试与实际数据对模型进行评价。

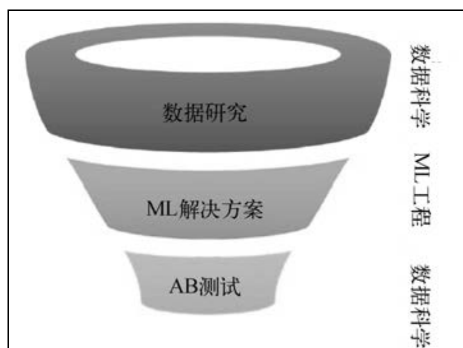



图11-1　从模型到产品

### 11.1.7　模型维护

另一个需要我们处理的问题是维护模型。模型是一成不变的吗？我们是对一个动态现象建模吗？它需要模型随着时间的推移调整预测吗？

模型通常在脱机批量训练中创建，然后对实时数据做预测，如图11-2所示。如果能收到对模型预测的反馈，比如股票是否如模型预测的那样上涨，某个候选人是否如预测的那样参选等，这些反馈应该用来进一步改善初始模型。

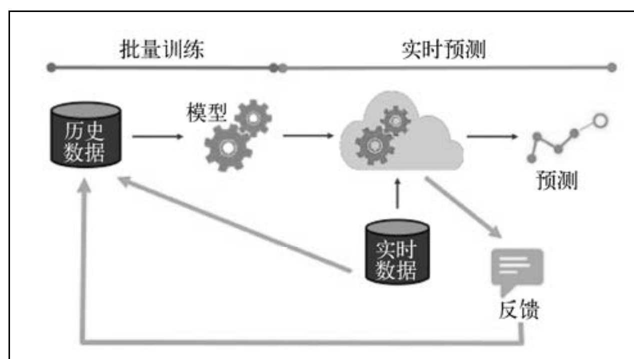



图11-2　创建模型与预测数据

对于改善初始模型，反馈的确很有用，但要注意你采样的数据。比如，如果有一个模型，用于预测谁会响应活动。最初你会使用一组随机联系的客户，他们带有特定响应或不响应分布和特征属性。模型只会把关注点放在客户的一个子集上，他们很可能做出响应，并且你的反馈会返回一个做回应的客户子集。通过包含这样的数据，模型会在特定子组上变得更准确，但在其他组上可能完全不对。我们将这个问题称为探索与利用（exploration versus exploitation）。可以在Ostugi等人（2005）与Bondu等人（2010）写的论文中找到这个问题的一些处理方法。

## 11.2　标准与标记语言

随着预测模型的应用越来越普遍，为了解决模型分享与建模过程规范化问题，需要有标准的模型开发流程与用于支持模型共享的交换文件格式。这一部分将介绍两个事实上的标准，一个涵盖数据科学过程，另一个指定一个交换格式，用于在不同应用中共享模型。

### 11.2.1　CRISP-DM

**跨行业数据挖掘标准流程（CRISP-DM）**描述的是一个数据挖掘流程，通常被工业领域的数据科学家采用。CRISP-DM将数据挖掘流程分为如下6个阶段：

- 商业理解
- 数据理解
- 数据准备
- 建模
- 评估
- 部署

图11-3中，箭头表示流程方向，它可以前后移动，经过各个阶段。并且，流程并不是在模型

部署之后就结束了。外部箭头表现了数据科学的周期性特征。流程中学到的知识可能引发新的问题，继续重复这个过程，对以前的结果进行改善。

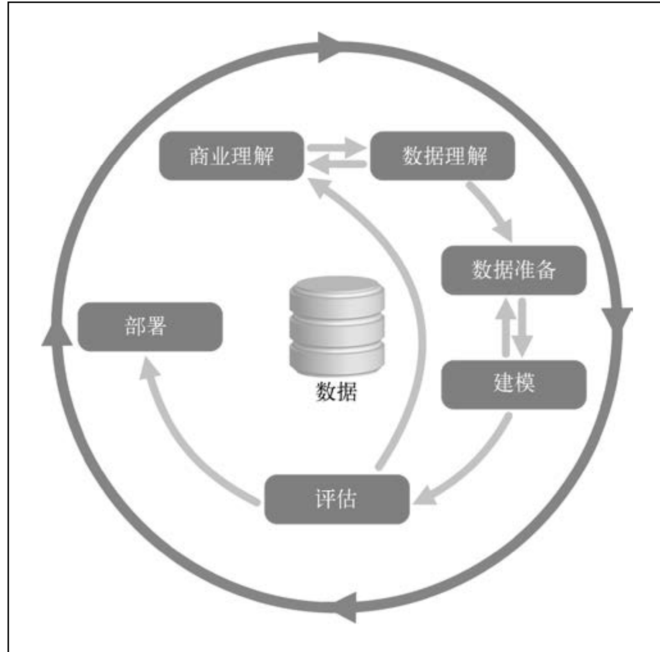



图11-3　数据挖掘流程

### 11.2.2　SEMMA 方法

另一个方法是“**采样、探索、调整、建模、评价**”（**SEMMA**）。SEMMA描述了数据科学中的主要建模任务，它不考虑业务方面，比如数据理解与部署。SEMMA由SAS Institute研发，它是统计软件的最大供应商之一，其目标是帮助他们的软件用户做数据挖掘的核心任务。

### 11.2.3　预测模型标记语言

**预测模型标记语言**（**PMML**）是一种基于XML的交换格式，它允许用户在不同应用与系统中轻松共享机器学习模型。PMML支持的模型有逻辑回归、神经网络、决策树、朴素贝叶斯、回归模型等。典型的PMML文件由如下几部分组成。

- 包含通用信息的头部
- 数据字典：描述数据类型
- 数据转换：指定标准化、离散化、聚合或自定义功能的步骤

- 模型定义：包含参数
- 挖掘模式：列出模型使用的属性
- 目标：允许对预测结果做后期处理
- 输出：列出要输出的字段及其他后处理步骤

PMML文件可以导入任何支持PMML语言的应用，比如Zementis公司的**ADAPA**（**Adaptive Decision and Predictive Analytics**）、**UPPI**（**Universal PMML Plug-in**）计分引擎、Weka（内建支持回归、广义回归、神经网络、**树模型**、**规则集模型**、**支持向量机模型**）、Spark（可导出k均值聚类、线性回归、岭回归、LASSO模型、二项逻辑回归、SVM）、Cascading（将PMML文件转换为一个Apache Hadoop上的应用）。

下一代PMML采用的是一种新格式，称为**"面向分析的可移植格式"**（**PFA**）。它提供了一个通用接口，用于部署跨环境的完整工作流。

## 11.3　云端机器学习

面对日益增长的数据，建立一个完整的机器学习栈很具挑战性。最近，**软件即服务**（**SaaS**）与**基础设施即服务**（**IaaS**）范式也深入到机器学习领域。现在的趋势是将实际数据处理、建模、预测转移到云环境，将主要精力放在建模任务上。

本节将介绍一些有前景的服务，它们提供了相关算法、预测模型（已经针对特定领域做过训练）与环境（增强数据科学团队之间的协同工作流）。

### 机器学习即服务

第一类是算法即服务，它们提供一些API或GUI，让你可以把预先编好的数据科学流水线组件连接起来。

- **Google Prediction API**：通过Web API引入预测服务的首批公司之一。这项服务集成在Google Cloud Storage中，用作数据存储。用户可以创建一个模型，并且调用API得到预测结果。
- **BigML**带有一个用户友好的图形界面，支持许多存储提供商（比如Amazon S3），并且提供各种数据处理工具、算法与强大的可视化功能。
- **Microsoft Azure Machine Learning**提供了机器学习算法库、数据处理功能以及图形用户界面，将这些组件连接成应用。另外，它提供了全管理服务，你可以使用它把自己的预测模型部署为可随时使用的web服务。
- **Amazon Machine Learning**进入市场相当晚。它的主要优点是可以与其他Amazon服务无缝集成，但算法数量与用户界面仍需进一步改善。

- **IBM Watson Analytics**：主要提供应用于特定领域的模型，比如语音识别、机器翻译、异常检测。其目标是工业领域，用于解决用户的特定问题。
- **Prediction.IO**是一个自托管的开源平台，它提供从数据存储到建模再到预测的全栈服务。Prediction.IO可以与Apache Spark通信，以便充分利用其学习算法。另外，它附带有各种针对特定领域的模型，比如推荐系统、流失预测等。

Predictive API是一个新兴领域，所以有名的例子不多。**KDnuggets**给出了50个机器学习API列表，网址如下：http://www.kdnuggets.com/2015/12/machine-learning-data-science-apis.html。

> 若想学习更多相关知识，请访问PAPI，网址为：http://www. papi.io，或者阅读Louis Dorard编写的*Bootstrapping Machine Learning*。

## 11.4　Web 资源与比赛

这一部分将介绍在哪些地方可以找到更多学习资源，共同参与讨论，进一步提升数据科学相关技能。

### 11.4.1　数据集

机器学习数据集最著名的存储库之一在加州大学欧文分校。UCI存储库包含超过300个数据集，涵盖各种领域，包括扑克、电影、葡萄酒品质、行为识别、股票、的士服务轨迹、广告等。每一种数据集通常都配有一篇研究论文，教你如何开始，并给出预测基准。

可以通过如下地址访问UCI机器学习存储库：https://archive.ics.uci.edu，如图11-4所示。

另一个有良好维护的数据集集合在GitHub，由Xiaming Chen维护：

https://github.com/caesar0301/awesome-public-datasets

这个Awesome Public Datasets存储库维护着400多个数据源链接，涵盖各种领域，比如农业、生物、经济、心理学、博物馆、交通等。数据集——特别是机器学习数据集——收录在图像处理、机器学习、数据挑战等版块下。

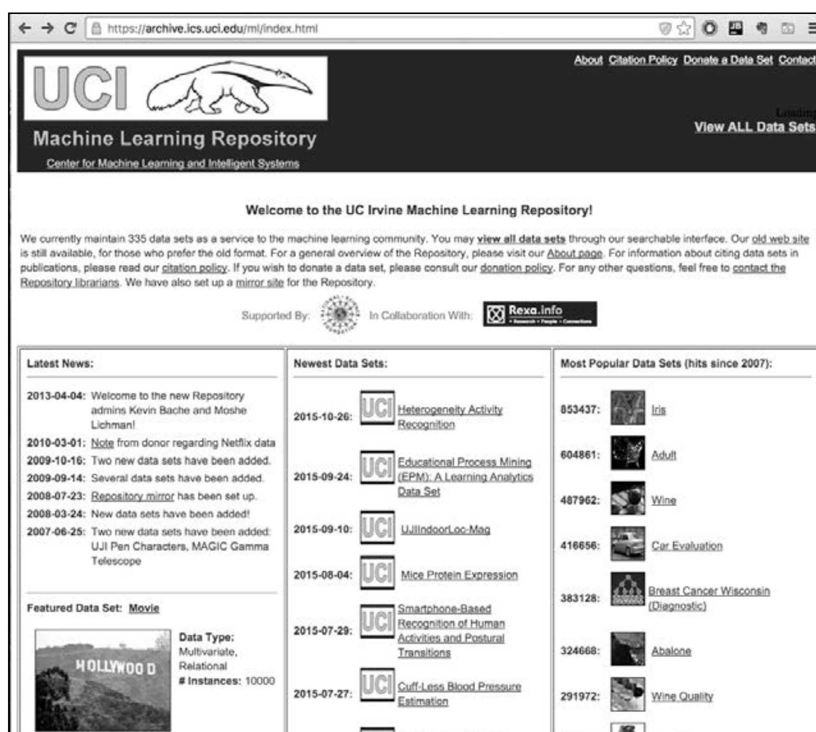



图11-4 UCI机器学习存储库

### 11.4.2 在线课程

网络上有大量优质课程，这些课程能够帮助我们尽快成为数据科学家。下面列出了一些免费在线课程，你可以通过这些课程学到不同知识与技能。

- Java在线学习课程
  - **Udemy：Learn Java Programming From Scratch**（https://www.udemy.com/learn-java-programming-from-scratch）
  - **Udemy: Java Tutorial for Complete Beginners**（https://www.udemy.com/java-tutorial）
  - **LearnJavaOnline.org: Interactive Java tutorial**（http://www.learnjavaonline.org/）
- 与机器学习有关的在线课程
  - **Coursera: Machine Learning (Stanford) by Andrew Ng**：讲授许多机器学习算法背后的数学，解释它们如何工作，探讨它们为何有意义（https://www.coursera.org/learn/machine-learning）。

- **Statistics 110 (Harvard) by Joe Biltzstein**：这门课程让你详细了解数据科学学习中经常听到的很多相关术语。可以在YouTube观看课程：http://projects.iq.harvard.edu/stat110/youtube。
- **Data Science CS109 (Harvard) by John A. Paulson**：这是一门动手课，你可以学到数据科学的Python库，也包括如何处理机器学习算法：http://cs109.github.io/2015/。

### 11.4.3　比赛

若想增加机器学习方面的知识，最好的方式是亲自动手解决实际问题。如果想积累一些“精彩”项目的经验，参加机器学习竞赛是个切实可行的起点。

- **Kaggle**：这是头号比赛平台，里面有各种挑战赛事，奖品丰厚，还有强大的数据科学社区和大量有用资源。可以通过如下网址访问：https://www.kaggle.com/。
- **CrowdANALYTIX**：这是一个众包数据分析服务，主要是生命科学与金融服务业，https://www.crowdanalytix.com。
- **DrivenData**：这是面向社会公益的数据科学比赛，网址为http://www.drivendata.org/。

### 11.4.4　网站与博客

除了在线课程与比赛之外，还有大量网站与博客刊登了数据科学的最新研究进展，以及解决不同问题的经验或最佳实践。下面列出的这些网站是很好的起点。

- **KDnuggets**：这是数据挖掘、分析、大数据、数据科学事实上的门户站点，涵盖新闻、故事、事件以及其他相关内容，http://www.kdnuggets.com/。
- **Machine learning mastery**：这是一个初级水平的博客，里面包含了许多切实可行的建议与指导，告诉你从哪里开始，http://machinelearningmastery.com/。
- **Data Science Central**：包含实用社区文章，涵盖多个主题、算法、商业案例，http://www.datasciencecentral.com/。
- **Data Mining Research**：由Sandro Saitta撰写的博客，网址为http://www.dataminingblog.com/。
- **Data Mining**：由Matthew Hurst撰写的博客，内容涉及文本挖掘、可视化与社交媒体，有很多有趣文章以及与web挖掘相关的主题，并且有许多应用被Bing与微软采用，http://datamining.typepad.com/data_mining/。
- **Geeking with Greg**：由Greg Linden撰写的博客，它是Amazon推荐引擎的编写者，也是互联网企业家，博客地址为http://glinden.blogspot.si/。
- **DSGuide**：这个站点收录了超过150个数据科学博客，网址为http://dsguide.biz/reader/sources。

### 11.4.5 场馆与会议

下面列出了几个顶级学术会议，涉及最新算法：

- **Knowledge Discovery in Databases (KDD)**
- **Computer Vision and Pattern Recognition (CVPR)**
- **Annual Conference on Neural Information Processing Systems (NIPS)**
- **International Conference on Machine Learning (ICML)**
- **IEEE International Conference on Data Mining (ICDM)**
- **International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp)**
- **International Joint Conference on Artificial Intelligence (IJCAI)**

还有如下一些商业会议：

- O'Reilly Strata Conference
- The Strata + Hadoop World Conferences
- Predictive Analytics World
- MLconf

此外，你还可以了解一下当地的聚会小组。

## 11.5 小结

本章是全书的最后一部分，讨论了模型部署的一些方面，并且了解了数据科学流程标准与可交换预测模型格式PMML。此外，还介绍了一些在线课程、比赛、Web资源与会议，充分利用这些资源将帮助你进一步掌握有关机器学习的知识。

我希望本书能够激励你更深入地了解数据科学，促使你亲自动手，尝试各种机器学习库，掌握解决各种问题的方法。请记得，书中所有源代码与额外资源都可以从本书的支持站点获取：http://www.machine-learning-in-java.com。

# 版 权 声 明



微信连接



回复“机器学习”“Java”查看相关书单



微博连接

关注@图灵教育 每日分享IT好书



QQ连接

图灵读者官方群I：218139230
图灵读者官方群II：164939616

**图灵社区**
**iTuring.cn**

在线出版，电子书，《码农》杂志，图灵访谈

- 快速了解用Java创建并实现机器学习
- 涵盖Mahout、Weka、Spark等常见库的功能和用法
- 介绍各种常见任务的机器学习应用，如基于数据库的预测预报、购物篮分析、图像识别、行为识别、文本分析等
- 在线支持网站提供书中所有示例代码以及其他入门资料：http://machine-learning-in-java.com





[PACKT] PUBLISHING

图灵社区：iTuring.cn
热线：(010)51095186转600

分类建议 计算机/机器学习

人民邮电出版社网址：www.ptpress.com.cn

ISBN 978-7-115-46680-8





ISBN 978-7-115-46680-8
定价：49.00元

# 看完了

如果您对本书内容有疑问，可发邮件至 contact@turingbook.com，会有编辑或作译者协助答疑。也可访问图灵社区，参与本书讨论。

如果是有关电子书的建议或问题，请联系专用客服邮箱：ebook@turingbook.com。

在这可以找到我们：

微博 @图灵教育：好书、活动每日播报

微博 @图灵社区：电子书和好文章的消息

微博 @图灵新知：图灵教育的科普小组

微信 图灵访谈：ituring_interview，讲述码农精彩人生

微信 图灵教育：turingbooks